Animation in Relational Information Visualization

by

Carsten Friedrich

Dipl. Inform. 1998

A thesis submitted to

The School of Information Technologies

The University of Sydney

for the degree of

DOCTOR OF PHILOSOPHY

March, 2002

I hereby certify that the work embodied in this thesis is the result of original research and has not been submitted for a higher degree to any other University or Institution.

Carsten Friedrich

Sydney

26th July 2002

I can dream about being Bugs Bunny, but when I wake up I’m Daffy.

(Chuck Jones)

Acknowledgements

I would like to thank my supervisor, Peter Eades, for his strong support throughout my degree. His scientific expertise, his encouragement, and his generous funding of my research have been essential for its success.

I would like to thank Michael Houle for his enthusiastic interest in my work. The extremely fruitful discussions with him lead to some of the most important results in this thesis.

Further, I would like to thank Martin Grajcar, Günther Fürthaler, Helen Purchase, Keith Nesbitt, François Bertault, and Sue Whitesides for many helpful discussions and insights.

My thanks go to the faculty, staff members, and my fellow graduate students at both the University of Newcastle and the University of Sydney for providing a friendly and enjoyable work environment. I also want to thank the InVision development team at DSTO for their valuable feedback, as well as for their generous financial support of my work.

I want to thank Rowena for her never ending and loving support throughout, as well as for her uncomplaining and repeated proof-reading of this thesis.

Finally, I am indebted to my family for their affection, support, and constant encouragement.

Contents

Abstract

In order to be able to navigate in the world without memorizing each detail, the human brain builds a mental map of its environment. The mental map is a distorted and abstracted representation of the real environment. Unimportant areas tend to be collapsed to a single entity while important landmarks are overemphasized. When working with visualizations of data we build a mental map of the data which is closely linked to the particular visualization. If the visualization changes significantly due to changes in the data or the way it is presented we loose the mental map and have to rebuild it from scratch. The purpose of the research underlying this thesis was to investigate and devise methods to create smooth transformations between visualizations of relational data which help users in maintaining or quickly updating their mental map.

CHAPTER 1

Introduction

The purpose of visualization is to communicate information, either between humans, or between computers and humans. Computers store and process information encoded as abstract numbers. Humans, however, prefer to perceive information in the form of two or three dimensional images. Visualization is the art of encoding data which is stored in a computer into images which make the semantics of the data easily accessible for humans. Information can either represent real world objects or abstract concepts. In the case of real world objects, we usually have a number of obvious and intuitive ways of visualizing the data. We can use a picture of the actual object or a more abstract drawing of the referenced concept, whichever is appropriate depending on the individual application. Figure 1.1 shows an example of visualizations of a real world object. The data we want to visualize is a car. Depending on the application, we can represent it with a photo of the car, as shown in figure 1.1.1, or in cases where details do not matter as much by a more abstract representation, as for example shown in figure 1.1.2.

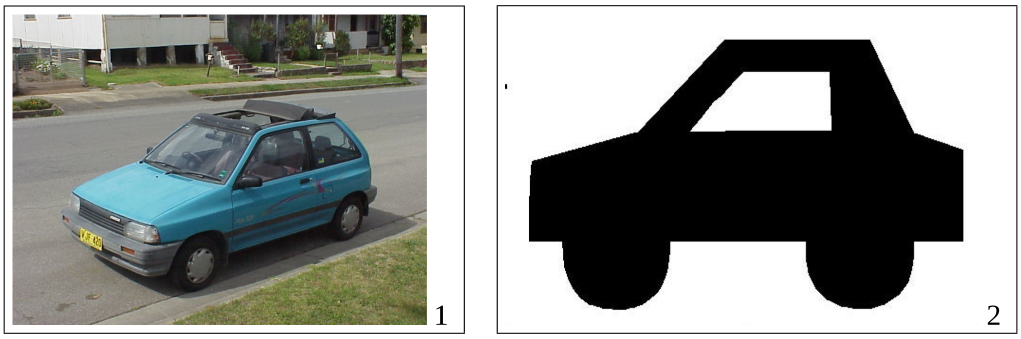


Figure 1.1: Data referencing real world objects can be visualized by displaying a graphical representation of the object. Depending on the application various degrees of abstraction are possible.

In the context of this thesis we are mainly interested in visualizations of abstract data. When dealing with purely abstract concepts we usually do not have an intuitive, obvious graphical representation that we can use to display the data. We rather have to encode the information by using a graphical *metaphor*. Figure 1.2 gives two examples for this case. In figure 1.2.1 we see graphical representations of human emotions; in figure 1.2.2 we see a graphical representation of the call-relationships between modules of a computer program. The modules are represented by boxes and the call-relationships between modules are visualized by lines connecting the boxes.

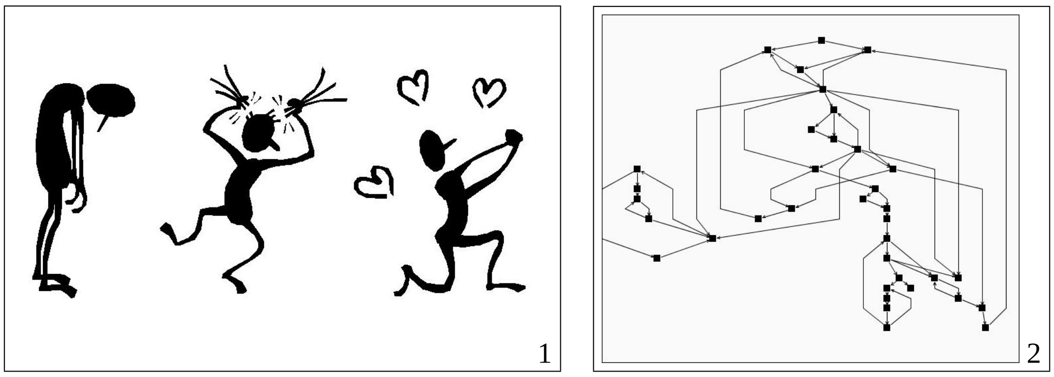


Figure 1.2: Data referencing abstract concepts such as emotions (1) or relations between data items (2) can be graphically displayed by using metaphors.

The options we have for creating a visualization do not only depend on the data, but also on the media on which we choose to create the visualization. The most basic devices for creating visualizations are pen and paper. They allow us to display data using arbitrary two-dimensional shapes which can be filled by a texture and positioned arbitrarily. This freedom is limited by the dimensions of the paper. In many situations, nowadays, we use computers and monitors rather than paper to display information. On a computer monitor we have one more degree of freedom: we can use the ability of a monitor to display images fast enough to create *animations*. An animation is a sequence of images that is characterized by subtle but highly structured changes between consecutive frames over space and over time. In the human brain these changes create the illusion of movement of the corresponding objects on the screen.

In this thesis we investigate methods for creating animations which improve the quality of visualizations of relational data in “dynamic environments”; this is, environments which contain data that is subject to modifications or data with changing graphical representations. Relational data is most commonly encoded in the form of a graph. A graph $G = (V, E)$ consists of a set of nodes V representing the data items and a set E of edges describing the relationships between the data items. In visualizations of graphs, each node is usually displayed as a graphical object such as a circle, rectangle, polygon, or an image. Edges are usually drawn as straight or curved line segments connecting the nodes of the graph. Figure 1.2.2 shows a typical example of a visualization of a graph.

In many applications graphs are not static but change their structure and layout according to user and application actions.

If nodes or edges are dynamically added to or removed from the graph, the drawing of the graph has to be updated. In order to fulfill certain aesthetic criteria, such modification can make it necessary to change the layout of large parts of the graph.

Similar problems arise in systems which are able to visualize and navigate in clustered graphs. Collapsing and expanding clusters can make it necessary to compute a new drawing to accommodate the changes.

Further, it is often desirable to generate different drawings, which emphasize particular aspects of the same underlying graph. Figure 1.3 shows an example. The graph represents dataflow in-

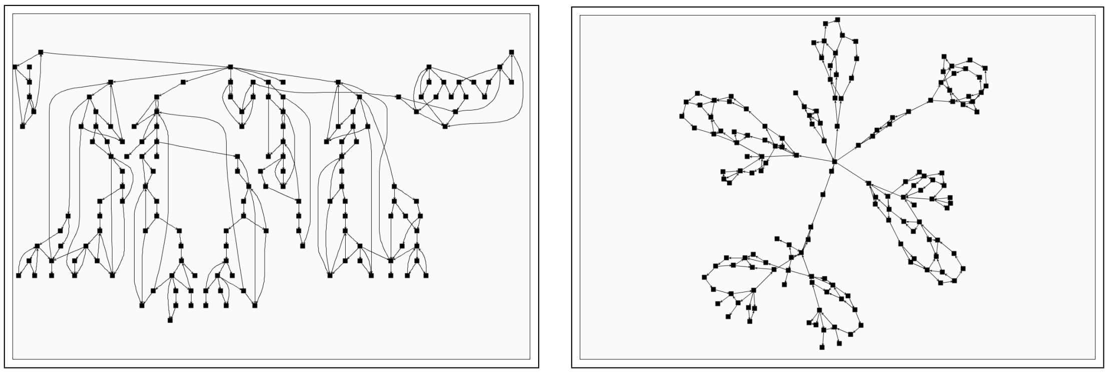


Figure 1.3: Two different drawings of the same graph. While the first drawing shows the hierarchical order of the nodes, the second drawing emphasizes how nodes are connected to each other.

formation in an intelligent telecommunication network. The drawing on the left side emphasizes the hierarchical structure of the dataflow. It is easy to see where data enters the system and how it proceeds downwards through the system. In contrast, the drawing on the right side emphasizes which parts of the network communicate with which other parts of the network. It is easy to identify tightly connected groups of nodes in network.

“Preserving the mental map” between changes to the drawings of graphs has been identified to be crucial for the usability of a system [8]. There are two possible approaches to this problem: either use graph drawing algorithms that try to minimize changes [13, 31, 32], or to communicate the changes in the form of an animation [43, 56, 119], that is, a smooth transition from the old drawing to the new drawing.

While specialized animation algorithms, for example [128], work quite well in practice, general animation techniques tend to fail to actually improve usability in many situations. Figure 1.4 shows an example of a bad animation[1].

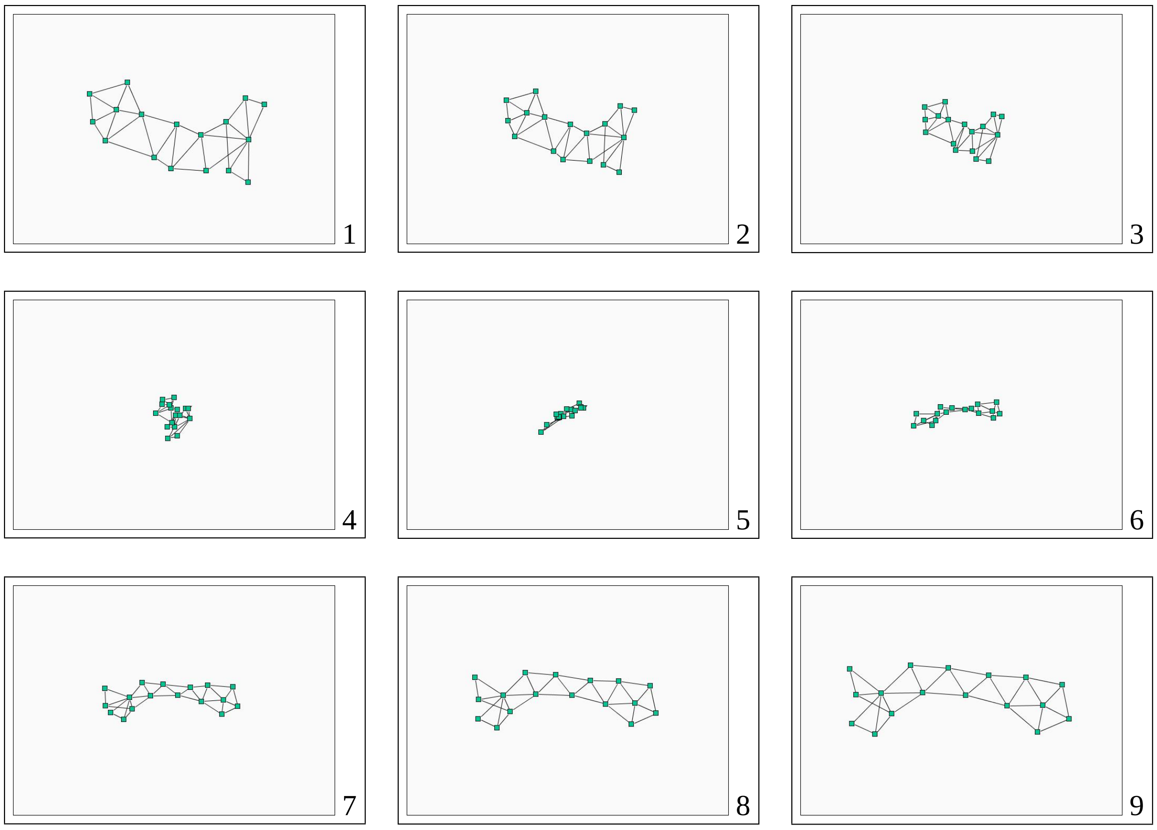


Figure 1.4: Example of a bad animation. The initial positions of the nodes are shown in image (1). Their target positions are shown in image (9). The remaining images show snapshots of a possible animation from the initial to the target drawing. In image (5) the graph is collapsed to a single point making it difficult to follow individual node movements.

An easier to follow, and therefore better, way of moving the nodes to their target positions is conceivable for this example. Figure 1.5 shows a possible better animation for the same initial and target drawings of the graph. We concern ourselves in this thesis mainly with investigating ways of computing good animations between visualizations of abstract relational data.

[1]All examples, in the form of Quicktime (TM) movies, are included on the CD attached to this thesis. If this thesis is viewed on a computer screen using an appropriate reader such as Acrobat Reader (TM) the examples can be watched by clicking on the image. Quicktime Player (TM) can be downloaded free of charge from `http://www.apple.com/quicktime/download/`; Acrobat Reader (TM) can be downloaded free of charge from `http://www.adobe.com/products/acrobat/readstep.html`

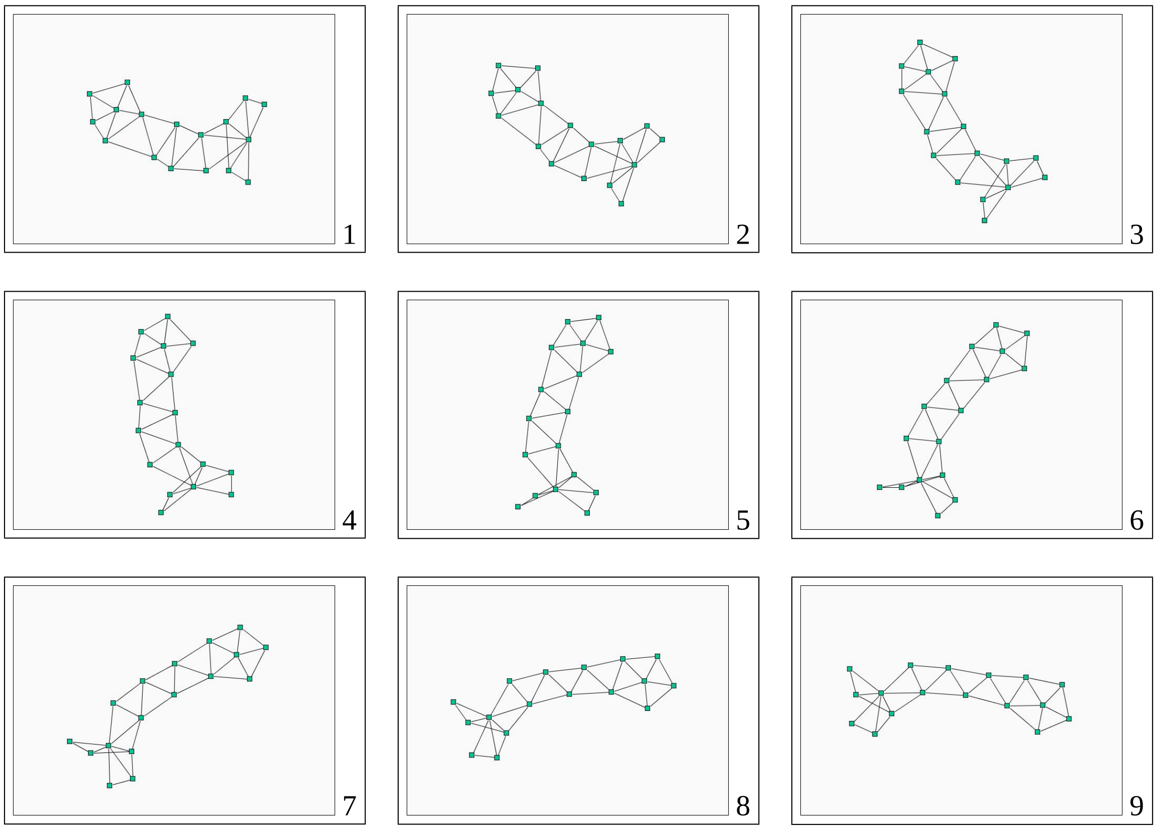


Figure 1.5: Example of a better animation for the same initial and target drawings as in figure 1.4. Instead of moving the nodes on a straight line from the initial to the target positions the nodes of the graph perform a rotation around the center of the graph. As the structure of the graph is maintained during the animation the movements are much easier to follow.

1.1 Contribution of this Thesis

No systematic research has been undertaken previously to investigate the use and properties of animations in relational information visualization. As the first attempt in this area, we introduce a general set of fundamental definitions which can be used to formally specify and evaluate animation methods for information visualization applications. Based on this framework of definitions we derive criteria and measures which enable us to compare the quality of different animation methods. We further propose and evaluate different algorithms for computing animations between visualizations of relational data. In this context we:

- Introduce a formal model describing animations.
- Introduce informal and formal criteria and measures for evaluating the quality of animations.
- Introduce a general framework for specifying and implementing animation methods.

- Present new approaches to automatically compute animations from given initial and target drawings of relational data. These approaches include:
  - an orthogonal interpolation method,
  - a classical force directed method,
  - a O(n log n) time force directed method,
  - an animation method that is based on linear regression analysis,
  - and a method based on cluster analysis techniques.
- We develop a benchmark suite for graph animation methods and evaluate the proposed animation methods, as well as existing animation methods, on the benchmark data.

The algorithms proposed in this thesis were implemented as a Java package and are publicly available for third parties to incorporate in their information visualization tools. DSTO Australia, which supported the research underlying this thesis, integrated the software in the InVision [84] information visualization tool.

Some of the results presented in this thesis have been published in [44, 45, 46].

1.2 Research Methodology

There is strong evidence that animation in user interfaces can help people to interact more efficiently with information visualization systems [10, 28]. There is further a common agreement on the conjecture that this is also true for the special case of systems which visualize dynamically changing relational information, and some graph drawing tools have actually implemented some naive animation methods. However, so far, no formal model for animations of relational information has existed and no comparative assessment of such animation methods has been performed. The fact that this is the first systematic work in the area of animations of relational information had several consequences for our research methodology.

We started by developing a basic formal model which allows us to formulate the problem of animating relational information visualizations, as well as possible solutions, in a systematic manner. In this context we identified a set of criteria which we believe is suitable for comparing the quality of different animation methods. These criteria are based on well established theories and empirical results in the areas of graph drawing (section 2.2), visual perception (section 2.4), and animation (section 2.3), as well as intuition and introspection of researchers in information visualization. A systematic evaluation and ranking of these criteria is a non-trivial problem and requires thorough HCI studies. The lack of precedent research in animations in relational information visualization, however, implies that we need to explore the solution space of the graph animation problem first. We nevertheless regard a systematic evaluation and ranking of quality criteria as a necessary step for further research in this area.

We derived mathematical formulations for the criteria which determine the quality of animations where possible. This allows an analytical discussion of proposed and existing animation methods. However, a purely analytical evaluation of animation methods is not without problems. Some of the methods proposed in this thesis are mathematically too complex to be analytically accessible. For the force directed animation methods in chapter 5, for example, the path of the nodes of the graph are only given indirectly as the solution to a set of non-linear differential equations which is not analytically solvable. It thus seemed necessary also to be able to evaluate the quality of animation methods empirically.

The lack of precedent research implies the lack of established test data for empirical work. Further, random data is clearly not suitable. We have thus developed a suit of benchmark tests. The benchmark suit is based on real life situations where animation might be used, as well as especially challenging animation problems. This benchmark suit is introduced in section 3.4.

The animation methods introduced in this thesis, in the majority of cases, try to explicitly optimize one or several quality criteria. Our evaluation methodology is thus as follows. We analytically discuss each method in respect to all criteria where possible. We further apply each method to our benchmark suit. During the benchmark tests we record the achieved values for our criteria. This enables us to discuss properties of animation methods which are not analytically accessible. By watching the animation and comparing the subjective impression to the recorded measures, we further gain insight in the adequacy and suitability of our set of criteria.

1.3 Structure of this Thesis

The thesis is organized in the following way:

- In chapter 2 we give an extensive overview of areas which are related to or underlying this thesis.

- In chapter 3 we introduce a model for graph animations which we subsequently use to formally describe animation methods. We also give a set of informal and formal criteria and measures which can be used to compare the quality of graph animation methods. We further define a suit of benchmark tests which allows an empirical evaluation of animation methods.

- In chapters 4, 5, and 6 we develop various methods of computing graph animations and evaluate the quality of these animation methods with regard to our measures and our benchmark suit.

- We close the thesis with a critical and comparative discussion of our results in chapter 7. This discussion leads to a set of guidelines for the use of animation methods in graph drawing environments. We also propose possible directions of future research.

CHAPTER 2

Background

In this chapter we introduce important areas which are underlying or related to this thesis. They form the foundation on which the thesis is built and from which it draws its motivation.

2.1 Visualization

“ A picture says a 1000 words ”

This popular proverb refers to the fact that humans prefer to perceive information in visual form. Using images, it is possible to communicate large amounts of data in a relatively short time. For example, we need much more time to read a description of a car, such as the one displayed in figure 2.1.1, than we need to get the same amount of information by looking at the picture. This effect is even stronger in figure 2.1.2. After looking at the picture of a person we are able to recognize this

Figure 2.1: A picture says a 1000 words. To communicate the same information in the form of a textual description we need much more time while achieving less accuracy.

person in a different context with high probability. To give a textual description of a person which allows us to recognize this person is almost impossible. In criminal investigations, for example, phantom images are created by experts from textual descriptions to make it easier for the public to identify possible suspects.

Visualizations are further used to communicate information in situations where no common language background exists, for examples on signs in international airports showing the way to customs, toilets, and ticket counters. A famous and rather unusual application of visualization is the attempt of the Planetary Society to send a message communicating information about the planet Earth to extra-terrestrial intelligent beings. As no common textual basis exists in this case, Carl Sagan, who designed the message tried to visualize as much information as possible assuming as little common background as possible. The result, in the form of a plaque which was attached to the Pioneer 10 and 11 spacecrafts, is shown in figure 2.2.

The aim of information visualization is to automatically transform abstract data into pictures. In contrast to the examples in figure 2.1 abstract information has no immediate non-textual visual equivalent. Examples of this kind of information are stock market data, call-relations between software modules in a program, social structures in a company, time-tables, and statistics of all kinds. To be able to visualize these types of data we have to use a graphical *metaphor*. That is, information has to be encoded using graphical objects which do not necessarily have an immediate relation to the data they represent. Figure 2.3 shows two examples where metaphors are used to visualize abstract data. The data items visualized in figure 2.3.1 are represented by the slices of a disc. The value of each data item corresponds to the size of its associated slice of the disc. It is very easy to judge the minimum, maximum, or variance of the data from the image. In figure 2.3.2 we see the visualization of the results of a chemical experiment. Two measuring methods are applied to various chemicals and the quality of each method is encoded using the height of a bar. It is very easy to see how the methods perform compared to each other and compared over various elements by looking at the visualization.

Designing metaphors is not an easy task. The visualization should correspond to the data in an intuitive way. Furthermore, the metaphor should support any "meta-information" that is to be communicated. Such meta-information can be how data items compare to each other, or how they are related.

We are accustomed to automatically categorizing objects by size. Encoding data values as pie-slices or bars, as we have seen in figure 2.3, is therefore a very intuitive and powerful method of visualization. However, which form of metaphor is appropriate in any individual application

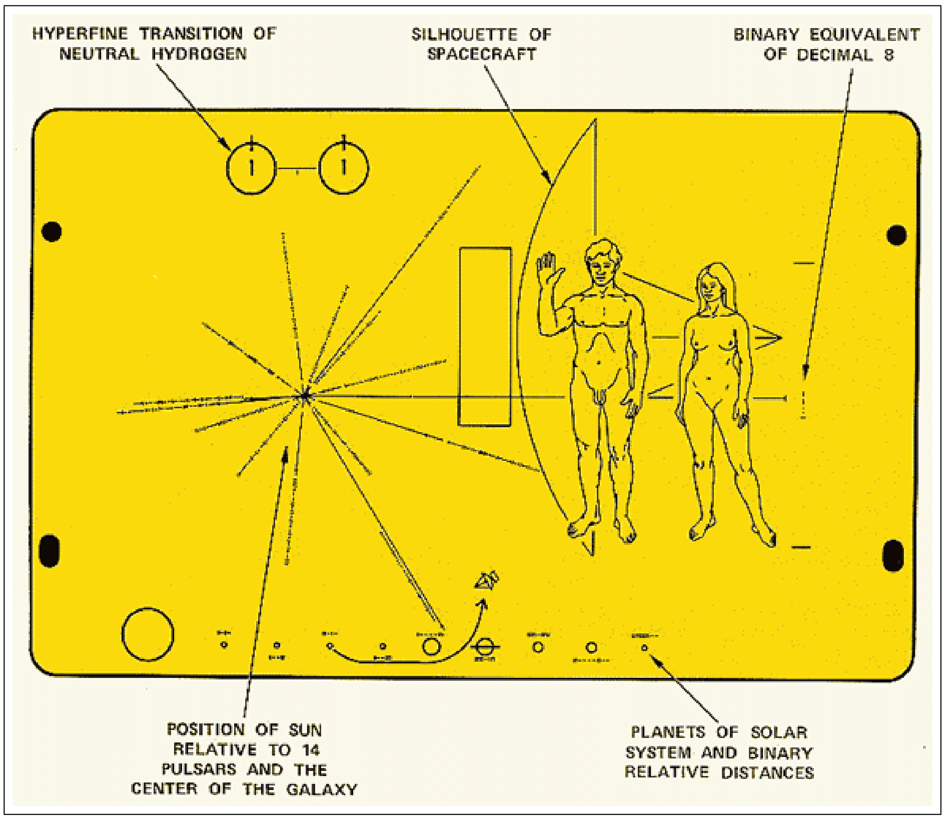
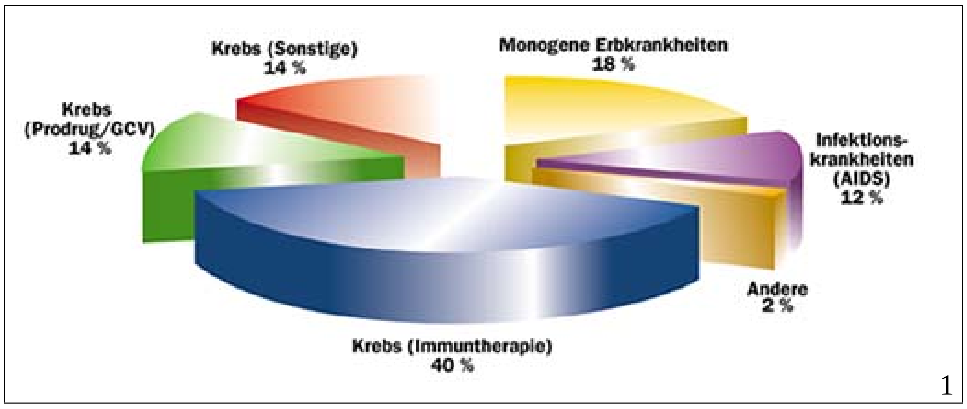

HYPERFINE TRANSITION OF NEUTRAL HYDROGEN
SILHOUETTE OF SPACECRAFT
BINARY EQUIVALENT OF DECIMAL 8
POSITION OF SUN RELATIVE TO 14 PULSARS AND THE CENTER OF THE GALAXY
PLANETS OF SOLAR SYSTEM AND BINARY RELATIVE DISTANCES

Figure 2.2: The Pioneer plaque was designed by Dr. Carl Sagan, one of the founders and first president of the Planetary Society, and drawn by his wife, Linda Salzman Sagan. On the plaque stand a human man and woman, the man's hand raised in a gesture of good will. The outline of the man and woman were determined from results of a computerized analysis of the average human. The key to translating the plaque is understanding the breakdown of hydrogen, the most common element in the universe. Hydrogen is illustrated in the upper left-hand corner of the plaque in schematic form and shows the hyperfine transition of neutral atomic hydrogen. A silhouette of the spacecraft, the planets in our solar system, and the position of our Sun in relation to 14 pulsars and the center of the galaxy are also illustrated on the plaque.
Caption: Planetary Society; Image courtesy of the Planetary Society

depends very much on the nature of the data displayed.

In figure 2.4 we see a visualization of documents on cancer research. Each individual document is represented by a green dot. Points which lie close together cover similar topics. It is easy to get an overview of the importance of certain topics, as well as to find documents which are similar to a given document.

In figure 2.5 we see a quite different way to visualize a data-set which is similar to the previous one. This time the focus is on the different topics covered in documents and how important they are. The importance of topics, measured by the number of documents dealing with each topic, is encoded by the height of mountains in a virtual landscape. Similar topics are located closer together than unrelated topics.



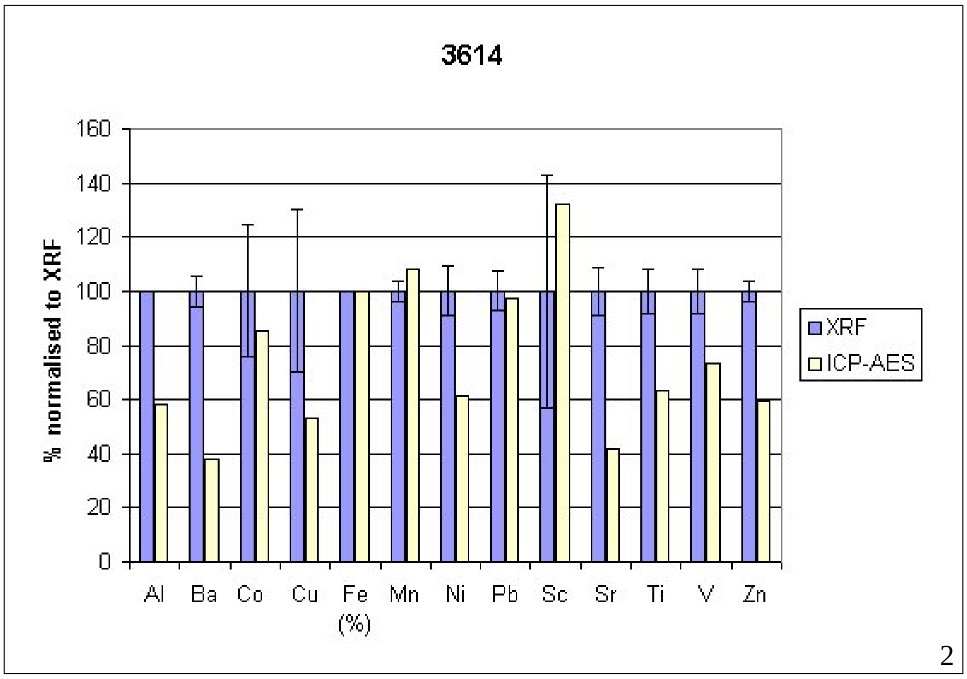
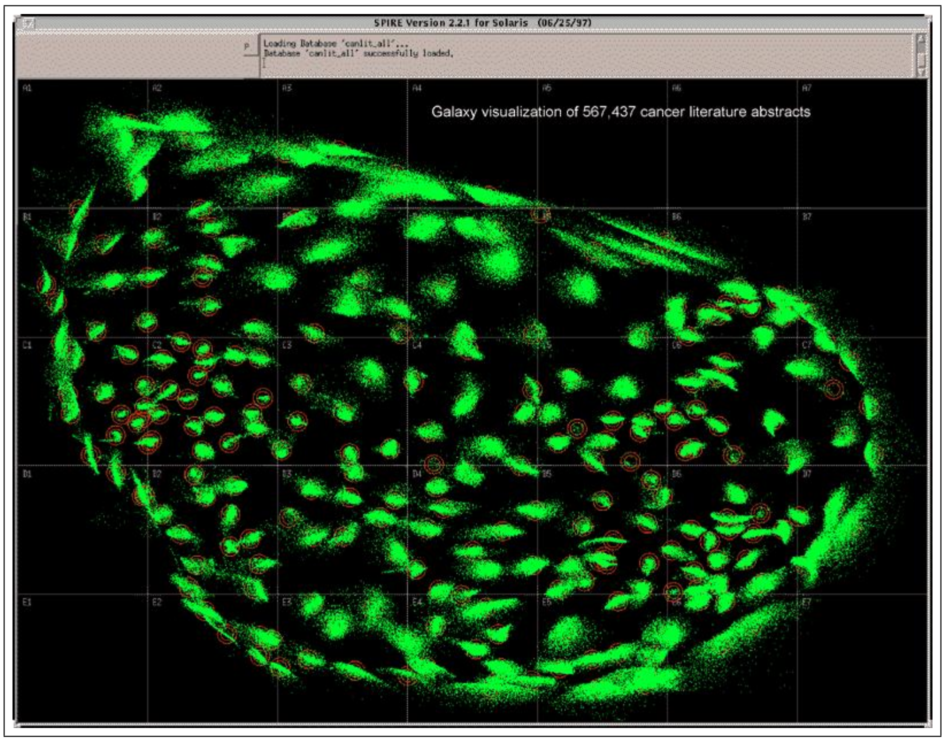


Figure 2.3: Metaphors are used to visualize abstract information. In image 1 we see proportions mapped to slice sizes of a disc. In image 2 we see a comparison of chemical methods applied to various elements. The length of the bars representing a method encodes its quality. Figure (2) courtesy of Rowena Mankelow

Sometimes different ways of visualizing the same data are used simultaneously in one picture. Figure 2.6 shows two examples.

Information visualization is a very well studied area with a long history of contributions. For a more detailed introduction we refer the reader to [22, 95, 115, 116, 117].

Several sub categories of information visualization, specializing in specific ways or applications, have been developed, and, in some cases, grown into areas of their own right. Among these are Distortion Techniques [93, 94], Software visualization [34], Algorithm Animation (Chapter 4, page 42), Network Visualization [9], n-D Visualization [38], and Graph Drawing [8].


SPIRE Version 2.2.1 for Solaris (06/25/97)
Galaxy visualization of 567,437 cancer literature abstracts

```
Loading Database 'canlit_all'...
Database 'canlit_all' successfully loaded.
```

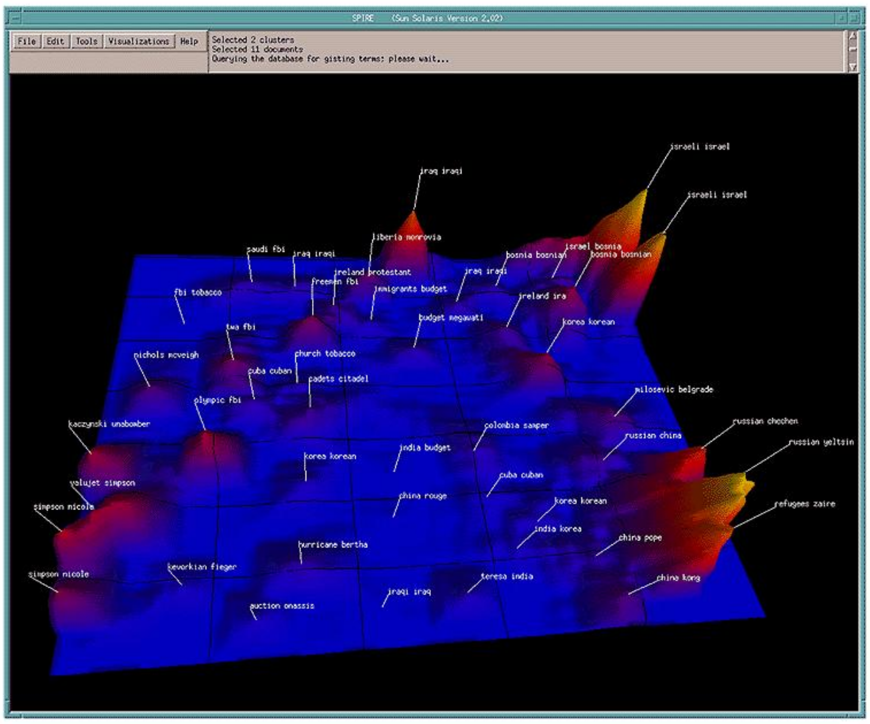
Figure 2.4: The Galaxies visualization uses the image of stars in the night sky to represent a set of documents. Each document is represented by a single *docustar*. Closely related documents cluster together while unrelated documents are separated by large distances. Courtesy of Pacific Northwest National Laboratory

2.2 Graph Drawing

The term *graph drawing* is used for visualizations of relational data. A graph $G = (V, E)$ consists of a set V of nodes and a list E of edges. Nodes represent data items and edges represent the relationships between the data items. Usually an edge $e \in E$ is defined by an unordered pair of nodes indicating a relationship between these nodes. Unless otherwise restricted, edges may occur more than once in a graph. Variations of this definition exist: In directed graphs, the order of the nodes in an edge is significant. In hyper-graphs, edges are defined by arbitrary subsets of the nodes of the graph.

Figure 2.7 shows an example of a graph. The nodes are displayed as circles and the edges are displayed as lines connecting two nodes. Graphs have many applications and are, for example, used to model software systems [34], represent knowledge maps [37], or to describe communication networks [9]. An in depth introduction to graph drawing and further references can be found in [7, 8, 61].

In many applications a graph does not contain specific geometric information. Theoretically,



Selected 2 clusters
Selected 11 documents
Querying the database for gisting terms: please wait...

Figure 2.5: In the ThemeView^{TM} visualization, the topics or themes within a set of documents are shown as a relief map of natural terrain. The mountains in the ThemeView^{TM} indicate dominant themes. The height of the peaks indicates the relative strengths of the topics in the document set. Similar themes appear close together, while unrelated themes are separated by larger distances. ThemeView^{TM} provides a visual overview of the major topics contained in a set of documents. Courtesy of Pacific Northwest National Laboratory

when displaying such a graph, the positions of the nodes on the screen can be chosen arbitrarily. For information visualization, however, the choice of positions of the nodes can make a large difference regarding the quality of the visualization.

Figure 2.8 displays the same graph as figure 2.7. This time however the nodes are positioned in such a way that no edges cross each other. This makes it much easier to perceive the structure of the graph. The aim of graph drawing is to produce “good” visualizations of graphs.

Many different ways of visualizing graphs exist, as for example shown in figure 2.9. In most cases however, nodes are drawn as geometric shapes and edges are drawn as lines. In this thesis we deal exclusively with graphs that are drawn in this standard way. Figures 2.10, 2.11, and 2.12 shows some typical examples of drawings of graphs.

The criteria for the quality of a drawing of a graph depend very much on the individual application in which the graph is used. Especially, as these applications are not necessarily limited to the area of information visualization. For example, if the nodes of the graph represent electronic

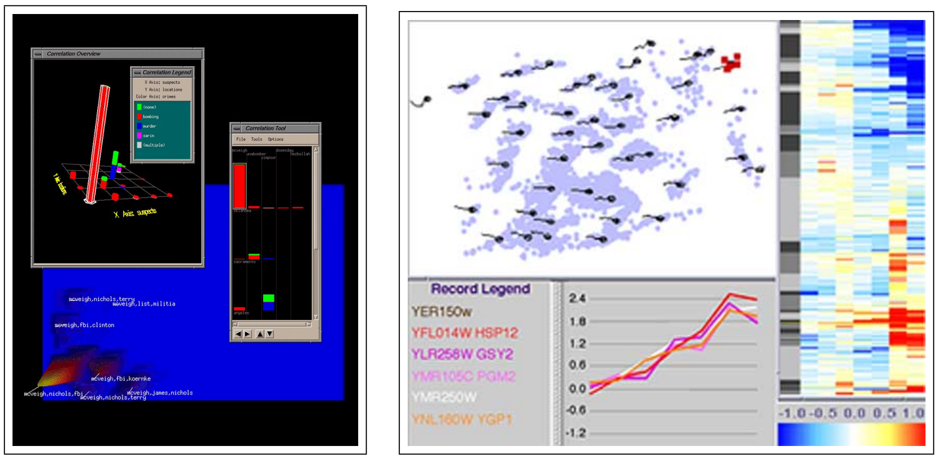


Figure 2.6: Different metaphors can be used simultaneously to visualize the same data in one picture. The different metaphors display different properties of the data. Courtesy of Pacific Northwest National Laboratory

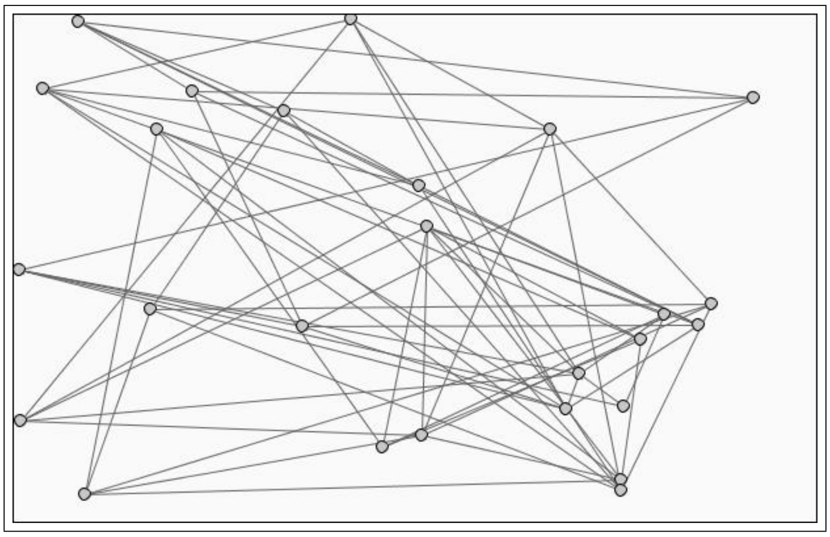


Figure 2.7: Example of a graph drawing. The nodes are displayed as circles and the edges are displayed as lines connecting nodes.

components and the edges represent the wiring of the components then a good drawing might be one that places the nodes and edges in such a way that as many nodes as possible can be placed on one chip without two wires crossing each other. In information visualization applications the main focus is to create drawings which communicate the information encoded in the graph as efficiently as possible. Several criteria have been identified to be essential in this case [8]:

- Minimize the number of edge crossings.

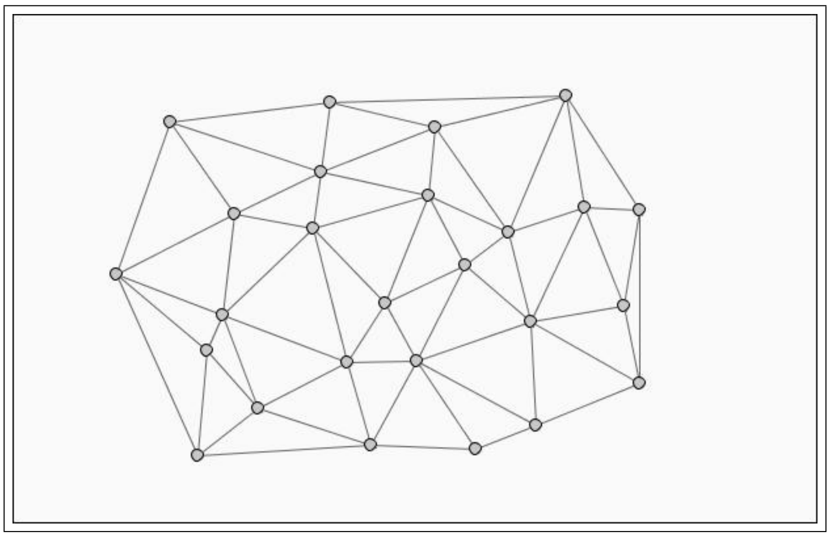


Figure 2.8: Example of a better drawing of the graph from figure 2.7

- Minimize the area of the drawing.
- Minimize the total edge length.
- Minimize the maximum edge length.
- Uniform edge length.
- Minimize the total number of bends.
- Minimize the maximum number of bends in an edge.
- Maximize the angular resolution of the edges.
- Maximize symmetry in the drawing.

In most cases not all of these criteria can be met in one drawing. Individual criteria might conflict or the computational complexity of optimizing all criteria in one drawing can be to high. Most graph drawing algorithms therefore try to optimize a subset of these criteria. Attempts have been made to classify these criteria according to how important they are for creating a good graph drawing [85, 86].

In some cases further restrictions to the drawing may apply. Edges may be allowed to only point into a specified range of directions, edges may not be allowed to cross each other, or node positions might be restricted to points on a grid.

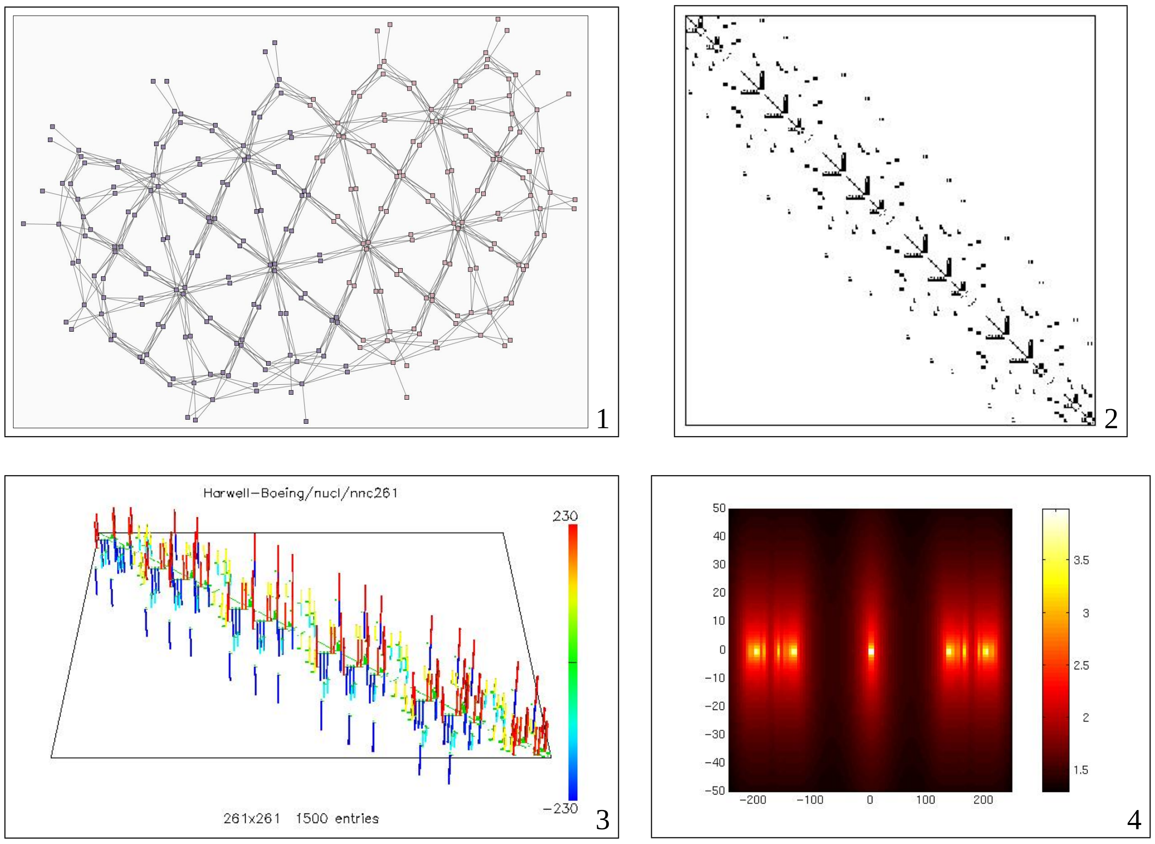


Figure 2.9: Examples for different ways of visualizing the same graph (NNC 261). The graph is courtesy of the National Nuclear Cooperation (UK) and derived from a model of an advanced gas cooled reactor core. Image (1) shows a traditional drawing of the graph. Image (2) shows a structure plot, image (3) a city plot, and image (4) shows the spectral portrait of the graph. Source: `http://math.nist.gov/MatrixMarket/data/Harwell-Boeing/nucl/nucl.html`

Many general and specialized algorithms for drawing graphs automatically have been developed and an extensive overview can be found in [7, 8]. Most prominent among the general algorithms are force directed approaches [29, 47, 59] and Sugiyama style approaches [33, 110, 111, 112].

2.2.1 Force Directed Graph Drawing

Force directed graph layout strategies compute node positions by simulating forces in a physical system. The nodes and edges of the graph are interpreted as physical components exerting forces on each other. While nodes repel each other, edges act as springs attracting nodes to each other. The force acting on each node $v \in V$ is given by:

$$\sum_{u \in V \setminus v} \frac{k_1}{d(u,v)^2} 1_{v,u} + \sum_{e=(v,u) \in E} \mid d(u,v) - l(e) \mid k_2 1_{v,u}$$

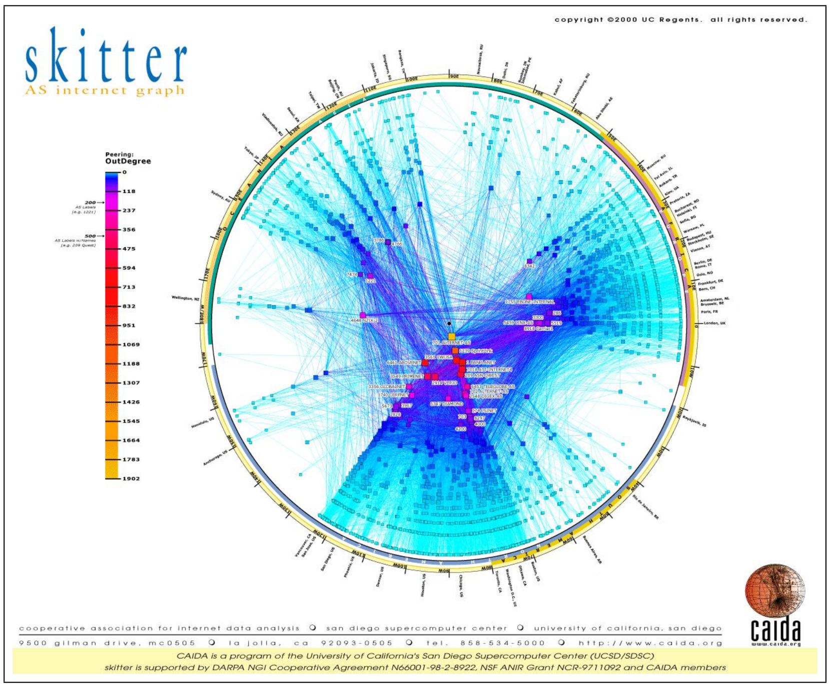

copyright ©2000 UC Regents. all rights reserved.
skitter
AS internet graph
Peering: OutDegree
0
118
237
356
475
594
713
832
951
1069
1188
1307
1426
1545
1664
1783
1902
cooperative association for internet data analysis san diego supercomputer center university of california, san diego
9500 gilman drive, mc0505 la jolla, ca 92093-0505 tel. 858-534-5000 http://www.caida.org
CAIDA is a program of the University of California's San Diego Supercomputer Center (UCSD/SDSC)
skitter is supported by DARPA NGI Cooperative Agreement N66001-98-2-8922, NSF ANIR Grant NCR-9711092 and CAIDA members
caida
www.caida.org

Figure 2.10: Example of a graph drawing. Image courtesy of Bradley Huffaker, CAIDA.

where $d(u, v)$ is the distance between node v and u, $1_{v,u}$ is the unit vector from v in direction of u, $l(e)$ is the optimal length of edge e, and k_1 and k_2 are constant factors.

Force directed methods minimize the energy, that is the sum of the forces, of the simulated system by moving nodes along their force vectors. Force directed methods can be used to generate drawings in two or three dimensions. Although this method is rather slow, using $O(|V|^2)$ runtime per iteration, and tends to run into local minima, it is very popular. The reason for this is that, despite its shortcomings, it tends to produce good drawings for many graphs and is comparatively easy to implement. Its property to emphasize symmetric structures in a graph can be identified as a further reason for its success. Figure 2.13 shows examples of drawings which were computed using a force directed paradigm.

Further, it is easy to extend force directed methods by adding additional forces to the system. For example Arne Frick et.al. [42] use a central gravity force to keep unconnected components of the graph from drifting apart.

The force directed optimization paradigm plays an important role in this thesis. Firstly, many of the most commonly used graph drawing algorithms are based on simulating force directed systems. We can therefore expect transitions between drawings that were generated using such algorithms to be one of the main applications of animations in relational information visualization. Seven of the

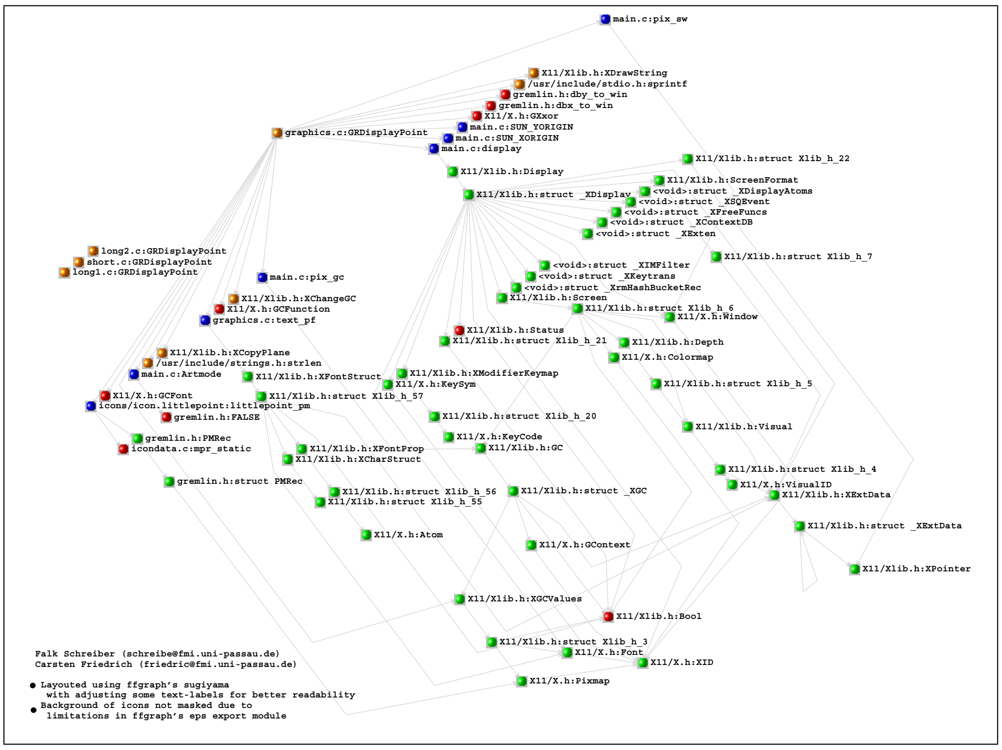


Figure 2.11: Example of a graph drawing. The graph represents dependencies within the X11 windows system.

ten scenarios in our benchmark suit involve at least one drawing which was generated using a force directed layout algorithm.

Secondly, we propose methods which use the force directed optimization paradigm to compute animations between arbitrarily generated graph drawings. These methods are presented in chapter 5 from page 146.

2.2.2 Sugiyama Style Graph Drawing

The Sugiyama style layout paradigm was initially designed as a method to produce hierarchical drawings of directed, acyclic graphs. It is however possible to extend the Sugiyama style approach to be able to handle general graphs and we discuss this briefly at the end of this section. Hierarchical drawings are characterized by the fact that all nodes and edge bends lie on parallel rows, called *layers*, and all edges are monotonic. Figure 2.14 shows some examples of graph drawings which were generated using Sugiyama style algorithms.

A drawing is produced in three steps where each step is based on and restricted by the result of the previous step. This can mean that a bad decision in an early phase can reduce the quality of

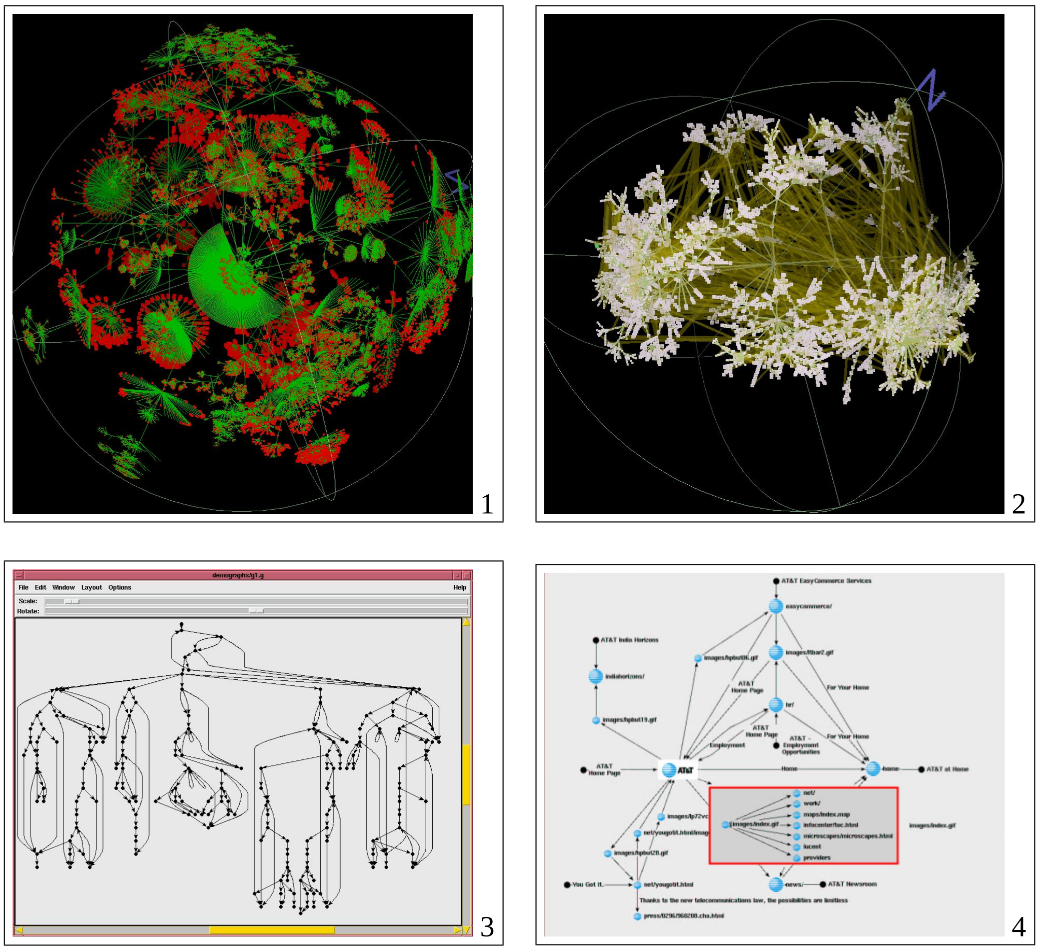


Figure 2.12: Examples of graph drawings. Images 1 and 2 courtesy of the Cooperative Association for Internet Data Analysis (CAIDA).

all subsequent phases. Furthermore theoretical problems underlying some of the phases have been proven to be NP-hard [50]. This, in combination with the strong interdependence of the phases make the choice of good and compatible heuristics essential. To produce a drawing the algorithms perform the following steps:

1. Assign nodes to layers

   In this phase each node of the graph is assigned to a layer. Each layer corresponds to a row in the drawing plane. The layer assignment has a strong influence on the area required by the drawing as it determines the height of the drawing and gives a lower bound on the width of the drawing. Many different approaches for this step exist [8]. Layer assignment methods usually try to optimize criteria such as minimal height or width of the final drawing. Minimizing both width and height in one drawing has been shown to be NP-hard [68].

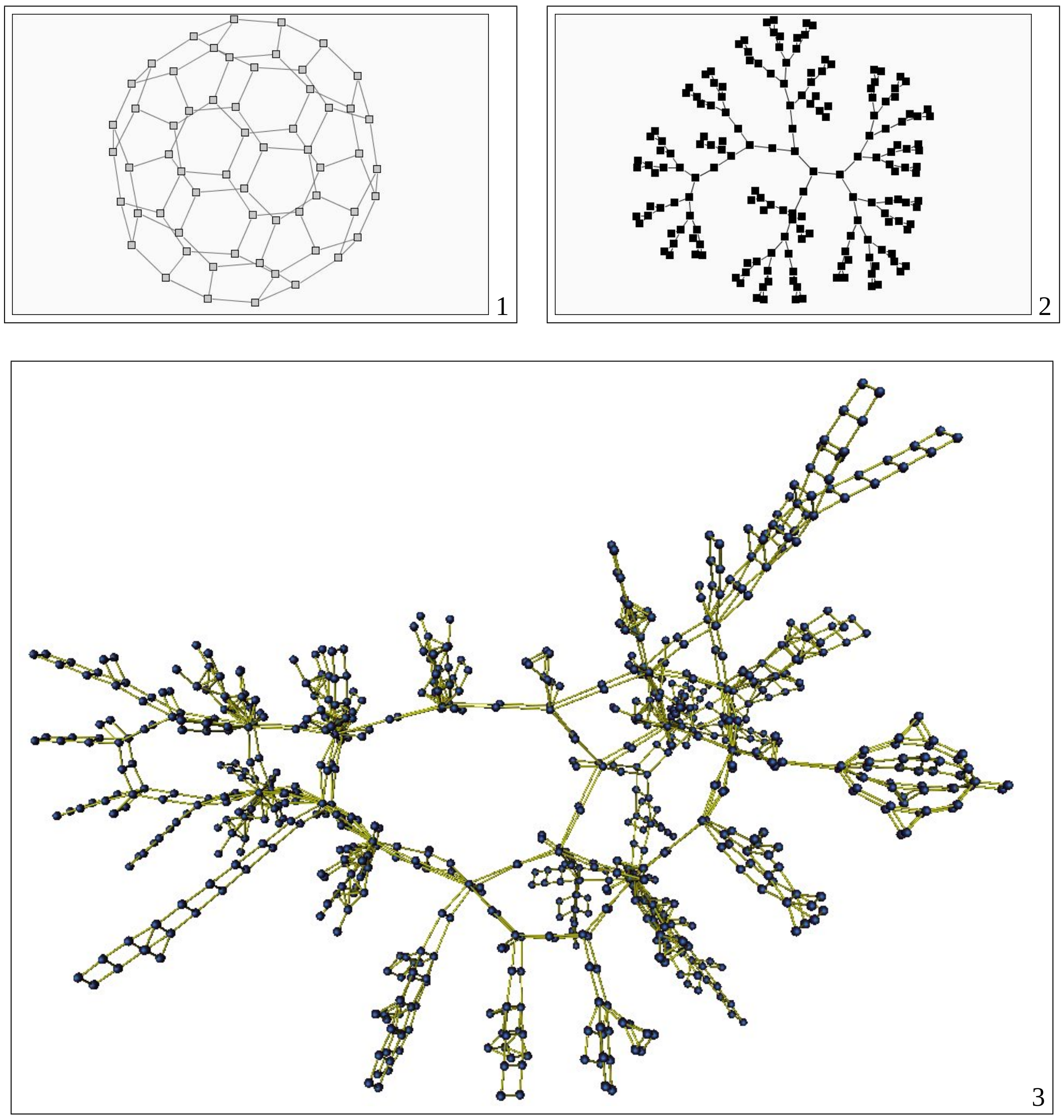


Figure 2.13: Examples of two and three dimensional drawings which were generated using force directed algorithms. Image 3 courtesy of Aaron Quigley.

2. Reduce edge crossings

   In this step the relative positions of the nodes within each layer are computed. The main aim in this phase is to minimize the number of edge crossings between two layers. This problem has been shown to be NP-hard [51]. Several good and fast heuristics exist [30, 58].

3. Assign final positions to nodes

   So far a vertical coordinate and a horizontal ordering has been computed for all nodes. In the

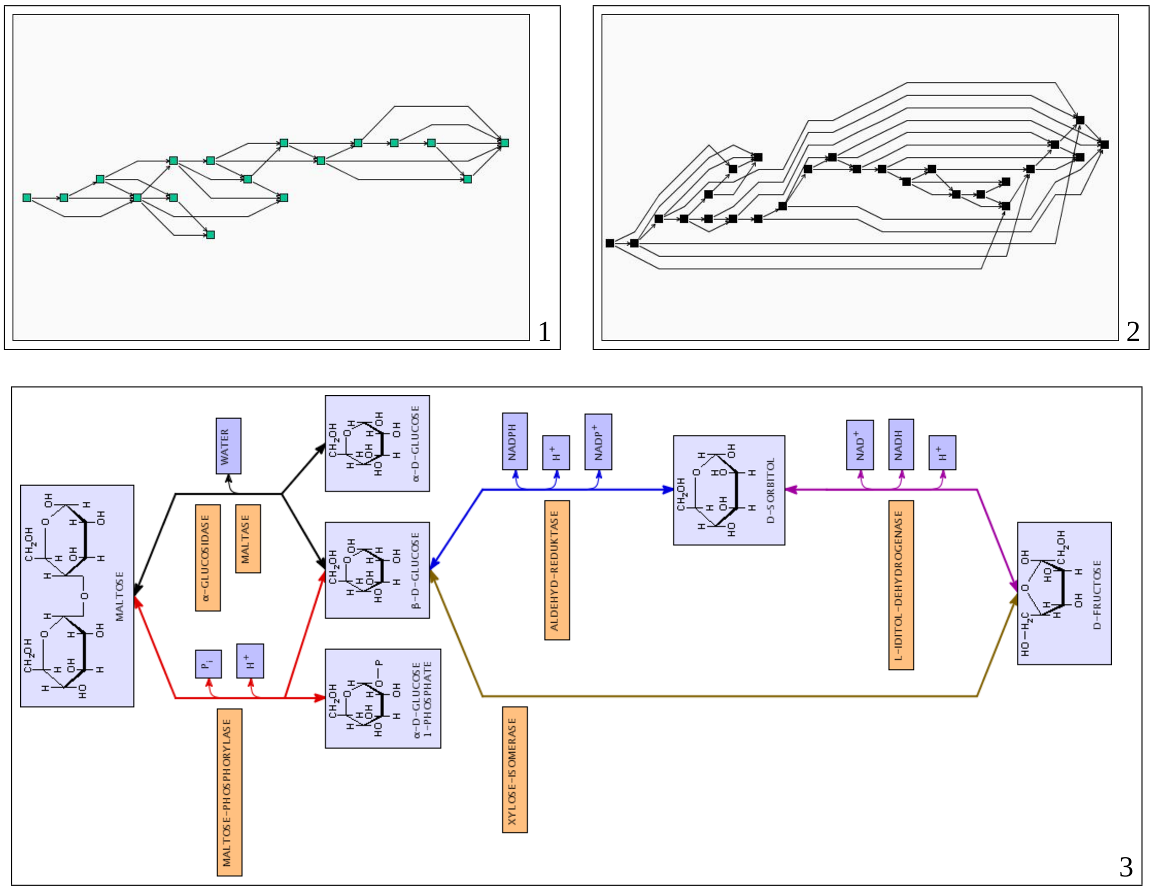


Figure 2.14: Examples of graph drawings produced using Sugiyama style layout algorithms. The images are rotated by 90 degree to better fit onto the page. Image (3) courtesy of Falk Schreiber.

last step the horizontal position for each node is determined according to various aesthetic criteria. Important criteria include that dummy nodes of the same edge should be placed directly above each other to form a straight line and that nodes should be centered over their successors. The theoretical problems underlying the optimization of some of these criteria have been proven to be NP-hard [8, 50, page 218]

The original Sugiyama style layout algorithm can be easily modified to handle general graphs. In the case of undirected graphs an arbitrary direction can be assigned to each edge of the graph as long as no cycles are introduced.

If the graph is directed but contains cycles the cycles can be removed by reverting individual edges. Although this can be done in linear time using a modified depth first search algorithm, trying to make a graph acyclic by reverting as few edges as possible has been proven to be NP-hard [50, page 192].

Graph drawing algorithms which are based on the Sugiyama method are most commonly used for displaying hierarchical relational information. We can therefore expect such drawings to frequently occur in graph animation applications. Consequently, several of the benchmark tests which are used in this thesis contain at least one drawing that was generated using a Sugiyama style graph drawing algorithm.

2.3 Animation

A movie consists of a sequence of still images. These images are called *frames*. This sequence is characterized by subtle but highly structured changes between consecutive frames over space and over time. The changes are perceived by the human brain as movements of the corresponding objects in the image. To create a live-action film a camera shoots consecutive pictures at a rate of approximately 25 frames per second. When these pictures are shown to humans using an appropriate projector they induce the effect of displaying continuous motion.

An *animation* could be characterized by the fact that the individual frames are not the result of a live-action scene recorded by a camera. Rather, each frame is drawn or modelled by a human or rendered on a computer. Although accurate, this characterization is by no means complete. It fails to specify the constraints on the individual images which determine whether a sequence of pictures is actually interpreted as moving or changing objects by a human spectator. Not all random sequences of images are perceived as continuous motion. It also focuses on the technical aspect of animation, ignoring its *nature* or *essence* which Norman McLaren tried to capture with his famous definition [80]:

> Animation is not the art of drawings that move but the art of movements that are drawn; What happens between each frame is much more important than what exists on each frame; Animation is therefore the art of manipulating the invisible interstices that lie between the frames.

Many further attempts to formulate a definition have been made, among others by Edward S. Small and Eugene Levinson [104], and Charles Solomon [105]. For a more in-depth discussion see [48]. The common denominator of all approaches however seems to be that in an animation the illusion of *"motion is created rather than recorded"*[48].

A large number of different techniques for creating the frames of an animation have been invented. These include *cutout animation*, *silhouette animation*, *Pixilation*, *Collage*, *Paint-on-glass*, clay and puppet scenes, *Rotoscoping*, *Cel-animation*, and computer rendered animations. A detailed discussion of these techniques can be found in [65]. The two techniques almost exclusively used for commercial and scientific animation nowadays are cel animation and computer generated animation; although some exceptions, such as the highly successful puppet animation movie *Chicken Run* (2000) by Peter Lord and Nick Park, can still be found from time to time.

Cel animation was invented independently and marketed by a joint company by Earl Hurd and Joseph R. Bray in 1914. The frames of the animation are drawn on several layers of transparent surfaces. In early times these transparents (called *cels*)were made of celluloid giving the technique its name. The biggest advantage of cel-animation is its cost effectiveness. For examples, in a sequence where a figure walks in front of a background, the background has to be drawn only once and can be reused in every frame in the scene. Only the changing parts have to be repainted in each frame. Further a complete motion cycle, for example a step, of a character can be reused repeatedly to animate an entire walking sequence. Cels can be used in more than two layers. If in a particular section only the eyes of a character move then three layers of cels are used: one for the background, one for the foreground character and one cel for the eyes of the character. Cel animation technique also allows the production of an animation by many concurrently working artists. Senior artists only sketch the main poses for each character and draw important *key frames*. The in-between frames are then drawn by assistant artists, or even computers.

In computer generated animations the frames of the animation are rendered by a computer program. Animation artists specify the scenery, objects, and actions of the animation using computer programs. The computer then uses these specifications to draw the frames. Modern computer programs allow more complex three-dimensional animations than a human artist could draw. Figure 2.15 shows an example of a computer generated animation.

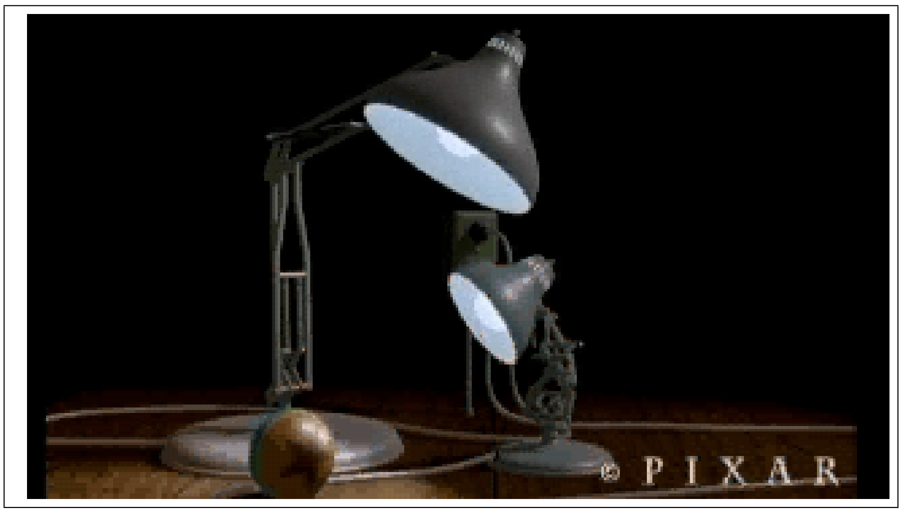


Figure 2.15: The movie "Luxo Jr." by John Lasseter is an early example of a completely computer generated animation. Source: `www.pixar.com`

The animation community is extensive and the number of works, especially by independent artists, is enormous. Computers and the internet provide the opportunity for artists to create and publish independent works at very low cost. Good resources for further research on current directions and events in the animation community are the *Animation Journal*, the Animation World Network[1], as well as the internet discussion groups *comp.graphics.animation* and *rec.arts.animation*. Excellent and extensive annotated bibliographies can be found in [63, 78].

2.3.1 Applications of Animations

Animations are used in many areas and for many differen purposes. In the following sections we briefly introduce the most important ones.

Create explicitly non-realistic characters and environments

This is probably the most prominent application of animation. The purpose usually is to create entertaining movies or film sequences. Examples are virtually infinite and, among many others, include cartoons featuring famous characters such as Bugs Bunny (Figure 2.16.1), Daffy Duck (Figure 2.16.2), and Walt Disney's Mickey Mouse (Figure 2.16.3).

They also include a long history of full feature-length movies from films such as *Snow White and the Seven Dwarfs* (Walt Disney, 1937, Figure 2.17) to recent hand drawn movies such as *The Emperor's New Groove* (Walt Disney, 2000) and *Princess Mononoke* (Miramax, 1999) by Hayao Miyazaki, or completely computer generated movies such as *Shrek* (DreamWorks, 2001, Figure 2.18) or *Final Fantasy: The Spirit Within* (Columbia Pictures, 2001, Figure 2.19).

Brush up or add elements to live-action scenes

Animations, especially computer generated animations, are a popular tool to postprocess real-life movie sequences. They are used to hide unwanted bits, to accentuate parts, or to add effects and objects to a scene. A prominent early example is the movie *Jason and the Argonauts* (1963) by Don Chaffey where fighting skeleton warriors were added to the movie using stop motion animation. In the movie *Forrest Gump* (1994) computer animations are used to add objects such as a falling feather, or a whole lake to some scenes. This is done in such a way that the viewer is not supposed to be aware of the animation and believes that what he sees is part of the live action scene. In *Star Wars: Episode I - The Phantom Menace* (1999) almost every scene was post-processed by a

[1]`http://www.awn.com/`

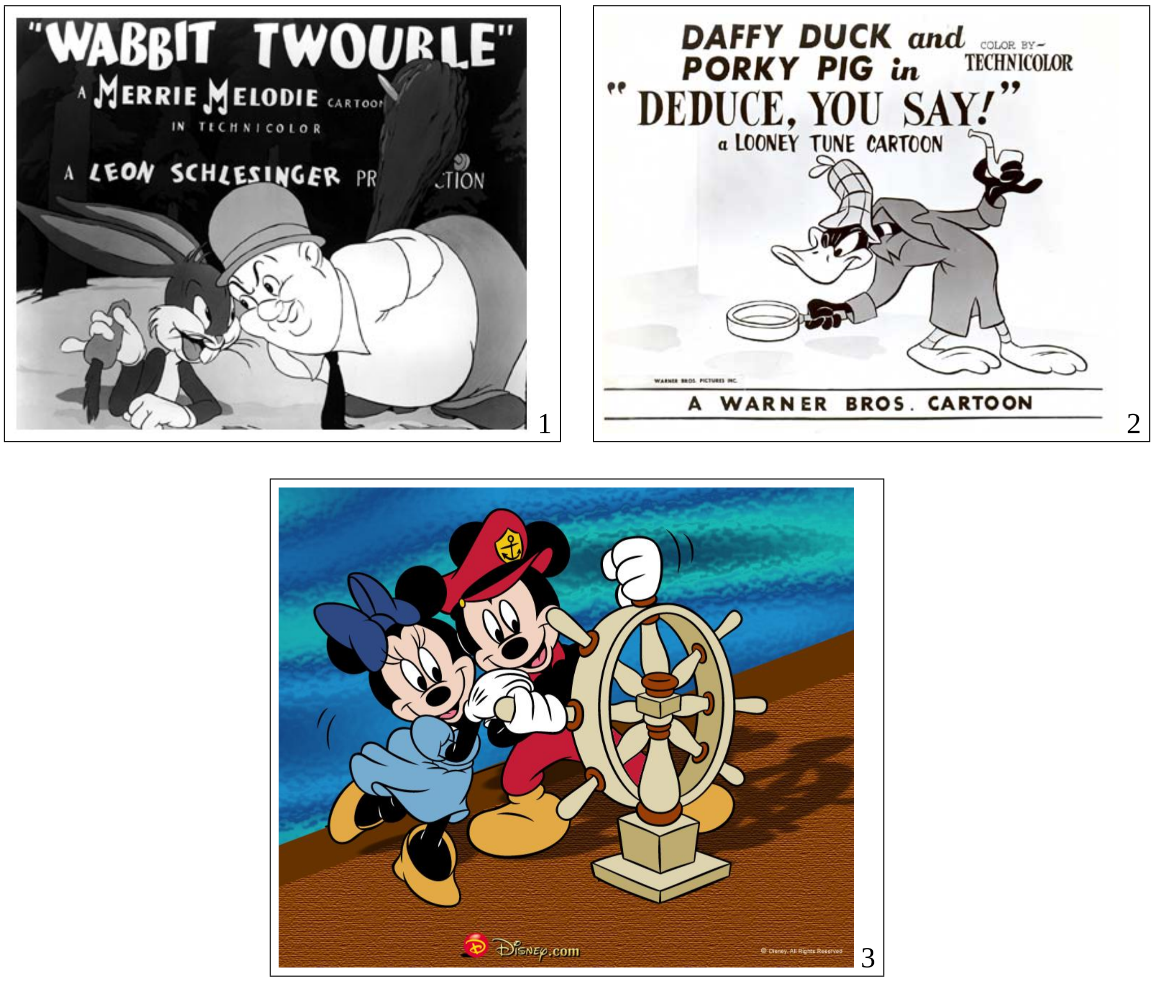


Figure 2.16: Examples of famous cartoon characters.

computer. Animations are also used to add photo-realistic content to scenes which would be difficult or impossible to photograph because the displayed objects no longer, or have never actually existed. Other prominent examples are the dinosaurs in *Jurassic Park* (1993, Figure 2.20) or all kinds of magical creatures in *Harry Potter* (2001). The movie *The Matrix* (1999) by Andy Wachowski and *Terminator 2: Judgment Day* (1991) by James Cameron make extensive use of computer animation to add special effects.

Create surreal elements or complete worlds within a real-world universe

Some directors facilitate the surreal atmosphere generated by animations as an explicit style element. The movie *Tron* (1982, Figure 2.21) by Steven Lisberger uses animation sequences to display the inside world of the computer into which the protagonist is transferred.

Figure 2.17: The movie “Snow White and the Seven Dwarfs” (1937) by Walt Disney was the first feature length animation film.

Figure 2.18: The movie “The Shrek” (2001) by Dreamworks is an example for a state of the art feature length completely computer rendered animation film. Source: `http://www.shrek.com/`.

Ralph Bakshi uses *rotoscoping* to display evil characters in the 1978 movie version of *The Lord of the Rings* (Figure 2.22). Rotoscoping is a technique where real actors are filmed in a live-action scenario. These recorded images are then modified by tracing and painting.

The 1988 movie *Who Framed Roger Rabbit* by Robert Zemeckis explicitly uses the contrast of live-action and animation universe as the framework of the story. A live-action detective switches between the real world and the cartoon world to help a cartoon rabbit to prove its innocence in a murder case. Further examples are *Mary Poppins* (Robert Stevenson, Walt Disney, 1964) or *Pete’s dragon* (Don Chaffey, Walt Disney, 1977).

Figure 2.19: The movie “Final Fantasy, The Spirit Within” (2001) by Hironobu Sakaguchi is another example for a state of the art feature length animation film which is completely computer rendered. Source: `http://www.finalfantasy.com/`.

Illustrate difficult or impossible to photograph or abstract information

Animations are not exclusively used for entertainment. In certain medical applications doctors want to look at parts of the body which are not accessible by a camera. Methods such as MRI or CAT-scan are used to gather data about specific regions of the body. One way to help a doctor in using this data for a diagnosis is to transform it into a visual representation of the region and generating an animation which presents a virtual fly-through. Another application is to simulate an operation before actually performing it. Some medical operations are extremely complicated and sensitive to mistakes. In such cases the possibility to simulate the operation before actually performing it can significantly increase the chance of success. Animations have further been used by lawyers before court to demonstrate possible events during a crime or an accident as, for example, reported in [57, 121, 129].

Animations are also very popular in education, and probably everyone has seen numerous animated films during their school time, explaining various physical laws, chemical reactions, or biological procedures. Figure 2.23 shows an example of an educational animation illustrating human hearing.

The benefits of animations in education, although they sound self-evident and students tend to

Figure 2.20: Screenshots from the movie “Jurasic Park’ by Spielberg.

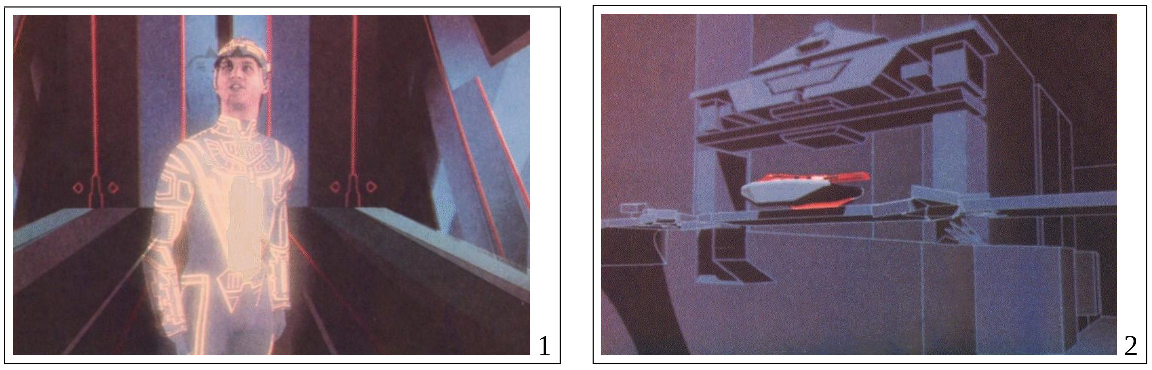


Figure 2.21: Screen shots from the 1982 Walt Disney movie *Tron*

have a generally very positive attitude towards them [107], are not undisputed. Positive evidence is reported when using animations in teaching contents based on physical objects, such as pumps, especially when accompanied by narrated explanations [73, 74]. In teaching abstract concepts such as algorithms the results are much less conclusive. Positive results are reported, among others, in [55, 64]. Little or no evidence for improvements in algorithm teaching by using animation are reported in [83, 107]. Although not conclusive, researchers found evidence that the learning situation in which the animation is presented [62] as well as the quality of the content of the animation [83] are key factors for the successful application of animation in teaching algorithms. For a more in-depth discussion of algorithm animations see section 4 on page 42.

Figure 2.22: Screenshots from the animation movie “The Lord of the Rings” by Ralph Bakshi.

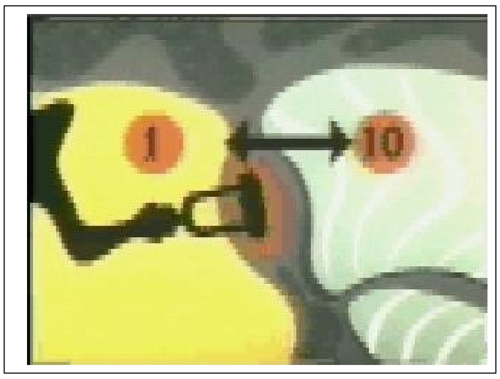


Figure 2.23: “Gateway To The Mind”, by Chuck Jones, 1958, Bell Science Series, Bell Telephone Labs. Animation showing anatomy of human hearing perception

Animation as an art form

The fact that this section is titled “Animation as an art form” should not be interpreted as an attempt to qualify the works and artists mentioned in this section as having higher artistic merit than the works discussed in other sections. Such a distinction is beyond the intention and abilities of the author. It is rather the case that most artists mentioned here have also been involved in commercial productions from time to time. We use this section to include forms of animation not covered before. These include mainly independent and non-commercial or scientific works. Some of these animations are produced using unusual techniques or materials, for example sand on glass, or Lego^{TM} bricks. Prominent artists and works using animation techniques include German born Oskar Fischinger (Figure 2.24), Karl Sims (Figure 2.25), David Brody (Figure 2.26), New Zealand born Len Lye (1901 - 1980, Figure 2.27), *anemic cinema* (1926) by French Dadaist Marcel Duchamp (1887-1968), Spanish Surrealist Salvator Dali who started the joint project *Destino* (Figure 2.28) with Walt Disney [2], French cubist Fernande Léger (1881-1955), German Dadaist Hans

[2]The project was never finished however.

Richter, Norman McLaren (1914 - 1987), John Halas (1912 - 1995) and his wife Joy Batchelor (1914 - 1991) who created the famous animation film *Animal farm*, computer artist Larry Cuba, Vibeke Sorensen, and many more.

Figure 2.24: Animation by Oskar Fischinger

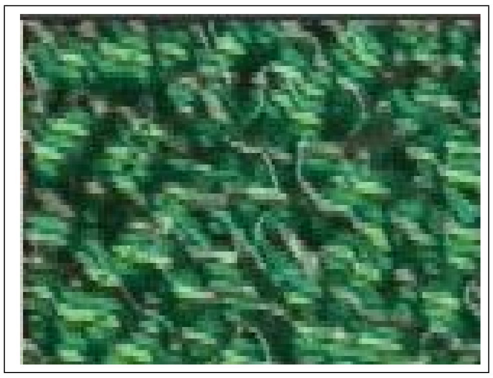


Figure 2.25: From ”Primordial Dance” By Karl Sims, 1991, Music By David Grimes, Visualized on Connection Machine CM-2

2.3.2 History of Animation

In this section we give a brief and rather incomplete overview of the historical development of commercial animation. A more in-depth discussion of this topic can be found in [12, 24] and on the web site `http://www.animationhistory.com/`.

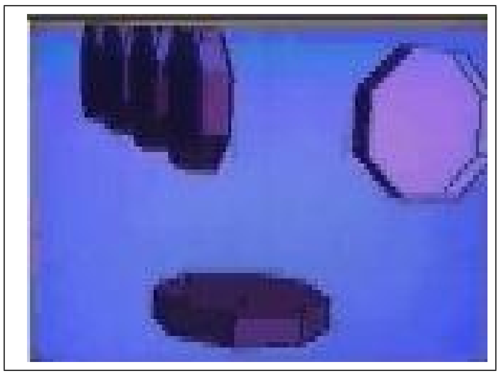


Figure 2.26: From *Beethoven Machinery* By David Brody, 1989, 16mm Film Vivace Beethoven String Quartet #16 in F Major Opus 135

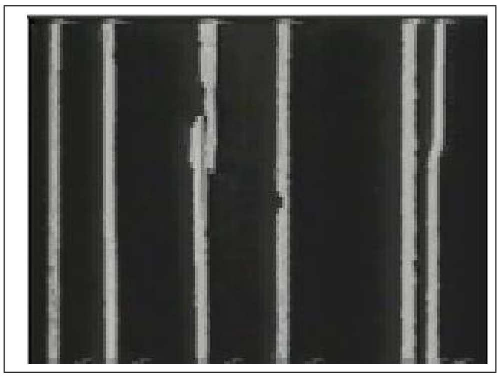


Figure 2.27: Excerpt from *Free radicals* by Len Ley, 1979, 16mm Film Music By Bridman Tribe of Africa

Optical devices to produce animations were invented long before film based motion pictures. The Thaumatrope dating back to the 16th century consists of a simple disc attached to two strings and a picture painted on both sides. Rotating the disc has the effect of superimposing the images, creating the illusion of one merged picture. This device does not produce an actual animation, that is the illusion of motion, but is based on the same principal as motion pictures, the *persistence of vision*. Persistence of vision refers to the phenomenon that the human eye continues to see images for a small period of time after they disappear. Due to this effect we are able to perceive individual pictures presented to us in consecutive order as a stream of continuous motion. Figure 2.29 shows the two sides of a Thaumatrope and how the images appear superimposed when rotated.

The Phenakistoscope invented in 1832 by Joseph Plateau, is based on the same principal and actually creates the illusion of motion. The motion sequence is drawn as a series of individual images on a disc. The disc also contains slots to synchronize the viewer with the pictures. When standing in front of a mirror and looking through the slots of the spinning disc the illusion of an

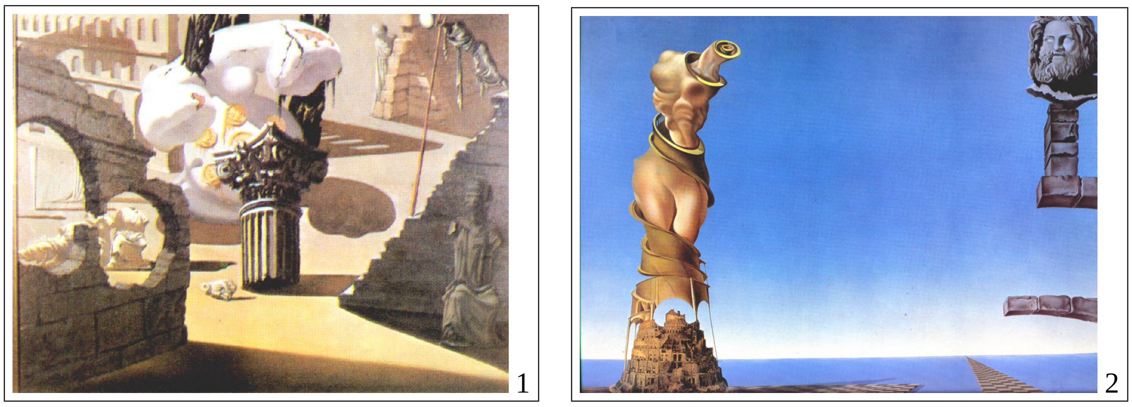


Figure 2.28: Drawings for the film *Destino* by Salvador Dali and Walt Disney. The project was never finished.

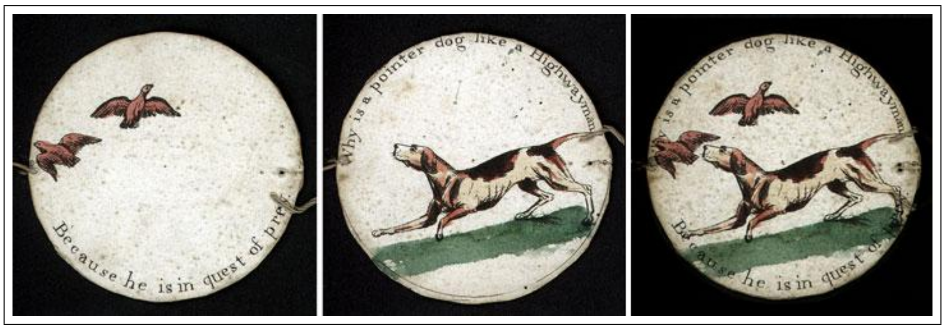

Because he is in quest of prey
Why is a pointer dog like a Highwayman

Figure 2.29: Thaumatrope. The first two images show the front and back of the disc. The last image shows how the images appear superimposed when rotating the disc.

animation is created. Further improvements on the same principal include the Zoetrope by William George Horner (1834) and the Praxinoscope by Emile Reynaud (1877).

Modern cinema was invented in 1895. Among the earliest animation artists working with projected films were Frenchmen Emile Cohl (1857-1938) creating *FANTASMAGORIE*, and George Méliès. In 1906 Stuart Blackton created an animated short film called *Humorous Phases of Funny Faces* which consisted of a series of comical faces and expressions drawn on a blackboard and recorded by a camera. With the start of World War I in 1914 European animation more or less completely disappeared and the animation film industry was almost exclusively dominated by American artists. In 1914 Winsor McCay created *Gertie the Dinosaur* which is considered to be the first animated movie featuring a main character and story. In the 1920s the animation market was dominated and driven forward by the competition of three companies: Fleischer Studios Inc., featuring characters such as Betty Boop and Popeye the Sailor; Walt Disney Productions, featuring Mickey Mouse, Donald Duck, etc.; and Pat Sullivan, producer of the famous *Felix the Cat* character by Otto Messmer. Success and failure of these companies was strongly related to their ability to invent and adopt new animation technologies. The demise of Pat Sullivan is commonly attributed to his refusal to use sound in cartoons such as *Felix the Cat*. Fleischer used sound as early as 1924 in *Song Car-Tunes* and Disney released *Steamboat Willie*, the first animation movie containing synchronized sound recorded on the film, in 1928. Further stages in this race include Disney's exclusive contract with Technicolor from 1932 to 1935, Fleischer's use of the Stereo-optical Process 1934, and Disney's first feature-length movie *Snow White and the Seven Dwarfs* 1937. In the early 1940s Fleischer had completely extended themselves financially and had to file for bankruptcy leaving Disney as the dominant force in the market.

Warner Bros. Cartoons which developed into another major competitor in the animation market was founded in 1930. Major animated characters featured by Warner include Porky Pig (1935), Daffy Duck (1937), Bugs Bunny (1940), Yosemite Sam (1944), Pepe Le Pew (1945), Tweety 1(947), Marvin the Martian (1953), and the Tasmanian Devil (1954). Disney won the first Academy Award for an animation film with *Flowers and Trees* in 1932. Metro-Goldwyn-Mayer (MGM) entered the animation stage in 1934 with *Happy Harmonies* and won their first Academy Award with *Yankee Doodle Mouse* in 1943. MGM featured characters such as Tom and Jerry 1940, Droopy and The Wolf, created 1943 by Tex Avery, *Fritz the Cat* 1972 by Ralph Bakshi, Thunderbirds (1968), and the Beatles' movie *Yellow Submarine*.

In the 1960s the production of animations started to develop in Japan and grew into a major player in the animation market. Early *anime*, as Japanese animations are commonly called, include *Tetsuwan Atom* (Atom boy, 1963) by Osamu Tezuka, *Science Team Gatchaman*, and *Space Cruiser Yamato*. Anime series such as *Dragon Balls* are a common part of every-day daytime TV since the 1990s. Recent major anime productions include *Neon Genesis Evangelion* (Gainax, 1995) and *Ghost in the Shell* (1995).

In America in the 1970s and early 1980s the animation market was dominated by low quality, mass-produced cartoons such as *Scooby Doo*, and *The Jetsons*. In the mid 1980s things changed for the better. Disney released the *Duck Tales* cartoon series in 1986 and started to produce new animated feature films such as *Who Framed Roger Rabbit* (1988) and *The Little Mermaid* (1989). Warner released *Tiny Toon Adventures* in 1989 and the *Batman* animation series in 1992. The 1990s also saw the advent of new players in the market such as Nickelodeon Network featuring popular animation series such as *Ren and Stimpy*, *Count Duckula*, and *Rugrats*. The highly successful animation series*The Simpsons* was started by 20th Century Fox Television in 1989.

As soon as computers with appropriate computing power were available in the 1960s people began using computer programs to create animations as well as using animations within computer programs. Ken Knowlton et al. started to create computer films in 1963 at Bell Telephone Laboratories. E. Zajac created *Two-gyro gravity-gradient attitude control system* which is considered to be the first computer animated film [70] in 1963. Stephen R. Russell is commonly regarded to be the first person to use animated graphics for the his interactive computer game *Spacewar* in 1961 [35]. Peter Foldes won the *Prix du Jury* at the Cannes Film Festival in 1974 for his computer film *La Faim*. John Lasseter received an Academy Award nomination for *Luxo Jr.* (Figure 2.15) and his short film *Tin Toy* was the first computer generated film to actually receive an Academy Award. Apple started to use basic animations such as opening and closing icons in their graphical user interfaces in 1984 [10]. Very elaborate animated user interfaces can be found on countless sites on the internet nowadays. *Toy Story* created by John Lasseter at Pixar Studios and Walt Disney Pictures in 1995 was the first completely computer generated feature-film. Dreamworks released their first computer animated feature film *Antz* in 1998 and the stop-motion feature *Chicken Run* in 2000.

2.3.3 Production Stages for Creating Animations

Although most animation studios have their own individual procedures to produce animations, creating a conventional cel-animation generally includes the following steps [70, pp. 4-5]:

1. Make up the story

   In the beginning of every animation is the story. The story is usually defined by:

   - The synopsis, a maximum one page summary.
   - The scenario, a detailed description of the story.
   - The storyboard, a comic strip version of the story, breaking it down into sequences of actions, scenes and shots. Exposure sheets are used to specify each animation shot down to cel-layers.

2. Define the layout

   In this phase the characters and shapes in the foreground as well as backgrounds are designed and their relations specified.

3. Produce the sound track

   In conventional animation productions the sound track is produced before the animation sequences. It is the responsibility of the animation artists to make sure that the animations conform to the sound track. Advanced productions even match lip movements of the characters to the spoken text.

4. Create detailed layout correlated with sound track.

5. Draw key frames

   Key frames are distinctive frames which define the main stages in a movement. As these frames mainly define the animation film, creative and usually senior animators are used for this task. In computed rendered animations key frames are modelled and movement paths defined using special software.

6. Draw in-betweens

   Assistant animators draw the frames that link the key frames. As the in-betweens differ only slightly from frame to frame and the start and endpoints are clearly defined by the key frames, less artistic expertise and creativity is needed for this task. Usually junior animators are assigned to these tasks.

7. Make a trial film / Pencil test

   So far all drawings are done with pencil and paper. These drawings are combined to create a trial film. Errors in the animation sequences can thus be identified and eliminated before further production stages are executed.

8. Xeroxing / inking frames

   In this stage the pencil drawn frames are transferred to cels. This is usually done by using cameras. The lines of the drawings are traced with ink.

9. Painting frames

   To produce colour animations the faces defined by the ink drawings have to be colored. Some simplified production techniques, for example using standard registered paper sheets or colour xeroxing, are sometimes used to skip the painting and inking phases.

10. Checking

    Final checks by the animation artists are necessary to ensure that the animation scenes are correct.

11. Recording

    The cels for each shot are mounted on an animation stand and recorded by a camera.

12. Postproduction

    Animation films like conventional films undergo several stages of post production steps such as developing the film, or editing and cutting to turn them into a final product.

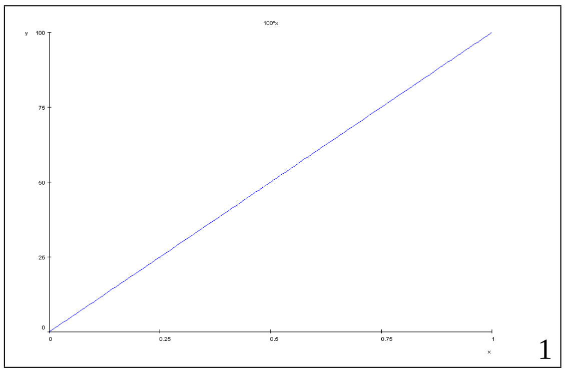
2.3.4 Computers and Animation

The ability of modern computers to generate high resolution images is nowadays widely used to assist in the creation of animation movies, online animations, and for creating interactive animations in user interfaces of computer programs.

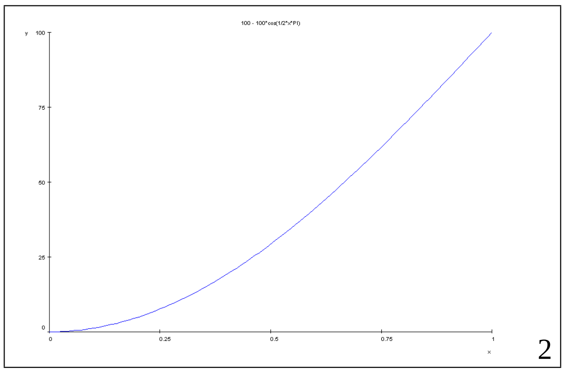
Computers are able to perform an ever increasing number of tasks involved in the various production stages of animation movies. Work which requires little creative skills such as xeroxing, inking, painting, and recording can be performed more or less autonomously by computer programs. Interpolation techniques can be used to automatically compute in-betweens if the transformation is specified accurately enough. The transformation has to describe the movement of each point of the drawing in space and time. The space component defines the path of the object in two or three dimensions. The speed of the point during the animation is defined by the time component. The speed is usually given by a continuous function describing either constant movement, acceleration, deceleration, or a combination of these. Figure 2.30 shows typical speed functions [70, page 49].

One of the key issues of this thesis is the introduction and discussion of new techniques for automatically finding transformations between key-frames when no explicit transformation is given.

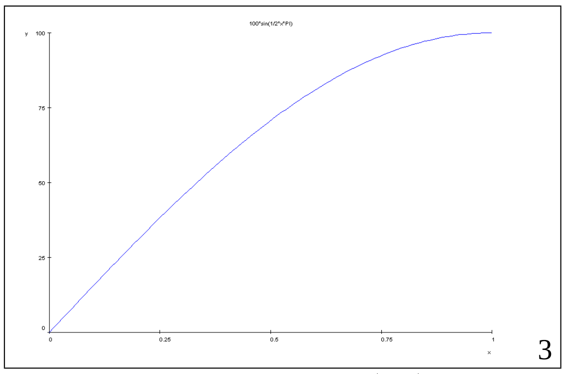
Highly creative tasks in the production of conventional animation movies, such as designing the layout, creating key frames, or composing sound tracks are still out of reach for computers. Although computers cannot perform these tasks, they nevertheless can still provide powerful tools to aid the animation artist. It is very difficult to create complex three dimensional animations without appropriate modelling software. Sophisticated software packages can assist the artist even further. Modern animation tools allow artists to automatically animate objects such as particles, swarms,



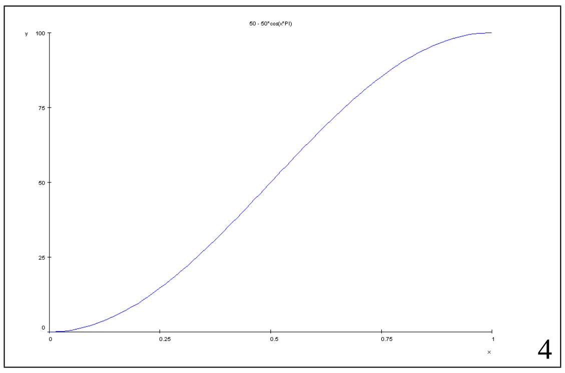
Constant: plx



Acceleration: $pl(1 - \cos(\pi \frac{x}{2}))$



Deceleration: $pl \sin(\pi \frac{x}{2})$



Acceleration and deceleration: $pl \frac{1-\cos(\pi x)}{2}$

Figure 2.30: Common speed functions for animations. The functions map a point in time (X-Axis) between two key frames (in fractions between 0 and 1) to a point on the path (Y-Axis). pl is the length of the path of a point.

and flocks of animals in a scene according to specified behavioral patterns [88, 89, 114]. Further, complicated motion sequences, for example human motions, can be using a motion capture device. The recorded motions can then be animated autonomously by a computer.

Animation in Information Visualization

In the context of information visualization, the amount of human intervention which is necessary to generate an animation is usually very small:

- Most applications do not need complex sound tracks. Specific sound effects or background music can easily be associated with events or objects and automatically be added by the software. They are part of the metaphor used to display the data.
- Object and background layout are defined by the metaphor used for the visualization.

The only steps of the animation production process which are relevant to create an animation in information visualization are:

- Story design,
- Key frame specification,
- In-betweening.

In the following sections we investigate what impact these three production steps have on the creation of animations in selected, prominent areas of information visualization.

Visualization of results of measurements and simulations

In many situations it is difficult or impossible to display all data resulting from a measurement or simulation in a single static image. We can distinguish between dynamic data containing a natural time component and static data which does not contain a time component. Data is considered to contain a time component if individual data items change during the time of measuring or simulating.

For data that contains a natural time dimension such as weather samples taken over a specific period of time or stresses on a car body during a crash test an animation is the most natural form of visualization.

CAT or MRI scans (which measure the distribution of water in a human body) and geological scans produce large amounts of static volume data. They do not possess a time component. Although the data itself does not change over time, a reasonable sized static image would not be able to adequately convey the information as only a fraction of the available data could be displayed.

Simulations are of a similar nature as measurements. The only difference is that the data does not result from recorded events of a real experiment but rather is computed according to a theoretical model. For the purpose of visualization this difference is irrelevant.

The outline of the story in these cases is defined by the general and specific nature of the data. In the case of static data the user usually wants to explore it by navigation. That is, the story is implicitly given by navigational commands of the user. Computer assistance is also possible in these cases. The computer could automatically identify suspicious data in an MRI scan and lead the user to the specific region. Key frames are also defined by the user navigation. In-betweens for natural navigational commands, simulating a continuous movement through the data, can easily be interpolated. If this is not intuitively possible, for example for jumps to specified locations, in-between strategies have to be defined and associated with appropriate navigation commands by the designer of the system. In the case of a jump the system could for example compute a feasible movement to this position and execute it or simply fade out at the current position and fade back in at the destination.

When animating visualizations of dynamically changing data the story and key frames of animation are predetermined by how the data changes. The computation of in-betweens can be arbitrarily complex and depends on the metaphors that are used to display data. For simple cases such as temperature changes which are displayed using a colour gradient, simple interpolation strategies can be used. In cases with complex metaphors or low granularity of the samples interpolation might only be possible with extensive additional information about the measured objects. For example the trajectory of a ball can be reconstructed and animated even if the granularity is poor by interpolating data samples according to the appropriate physical laws.

Computer games

In computer games the story is usually defined by three parties. The developers of the game engine define the general framework for the story. The story has to be told by the means provided and the restrictions imposed by the game engine. This can have severe consequences: a group of architects tried to use the 3D-engine of the first person shooter game *Quake*™ to generate virtual walks through buildings. They finally had to abandon this project as the game engine did not provide the possibility to use un-armed characters [71]. Game engines are complex computer programs and have to be designed and written by humans.

The second group involved in creating a computer game specifies the layout, objectives, and story underlying the game. They design the levels and maps of the game, the background music, sound effects, and objects and non player characters with whom the player interacts. The degree of freedom in this task depends very much on the flexibility of the game engine and varies widely from game to game. In many board games, such as chess programs, level design is restricted to the graphical design of the board and pieces. On the other extreme most MUDs[3] [103] allow level designers or even players to extend the semantics of the game by providing a programming interface. Usually levels are designed by humans. Computer assistance occurs in the form of drawing, image

[3]Multi User Dungeon games (MUDs) are games which allow many players to concurrently navigate and interact in a dynamic virtual world.

processing, and modelling software. Often game specific level editors exist which help the designer. In some cases the games provide functionality to randomize level design (Nethack) or create levels automatically according to design parameters. For example, in the computer game *Civilization*$^{\text{TM}}$ the computer can create maps and scenarios based on user specified parameters such as number of opponents, land-sea ratio, and climate.

The last party involved is the player. The player interactively navigates through the levels of the game and interacts with objects and non player characters thereby defining the remaining part of the story. The degree of freedom depends on the game engine and level design. It is difficult to do something with a chess program other than to try to set the opponent check mate. In modern role playing games such as *Baldur's Gate*$^{\text{TM}}$ player actions can have a significant impact on the story line.

Independent of the degree of freedom that is given to the player, the game engine has to be able to autonomously create the appropriate animation sequences for the user input. Usually this is achieved by a combination of behavior descriptions, simulation of physical laws and pre-designed animation sequences. In most cases the key frames are defined by the user actions and the game physics. The in-betweens are either interpolated, for example in modelled three dimensional levels, or consist of repeatedly displayed animation sequences which were pre-designed by the game designers.

Visualization of abstract data

Abstract data and concepts are often difficult to understand if presented in textual form. Visualizations are a valuable help to either try to interpret them or to communicate them to others. If the data contains a time component or is too big to be displayed in one static image, animations can significantly improve the quality of a visualization. Figure 2.31 shows two snapshots of a three-dimensional model derived from a chaotic system defined on coupled lattice maps. The user can navigate through the model to look at it from different points of view gaining a deeper understanding of the data.

Figure 2.32 shows an animation used to visualize a diffeomorphism related to the connection between the spin of fundamental particles and the geometry of space-time. It shows the behavior over a 720^o rotation around the center.

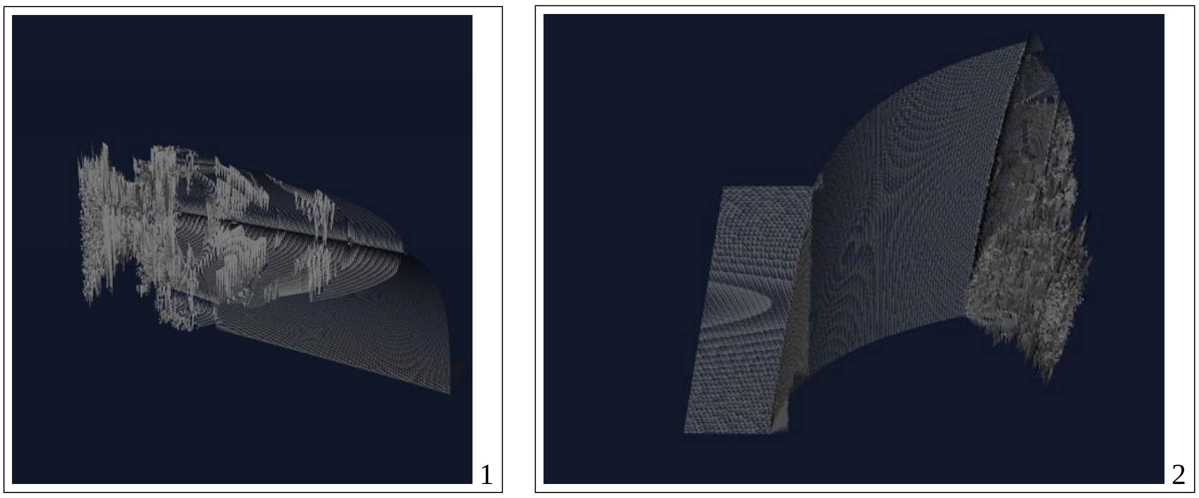


Figure 2.31: Model used in the visualization of a chaotic system defined on coupled lattice maps. Courtesy of C. Ormerod, Prof. B. Pailthorpe, N. Bordes School of Physics, University of Sydney

Algorithm animation In computer science education, teaching complex algorithms such as sorting, shortest path, or fast Fourier transformations to students is a very challenging task. When only describing the algorithm or just presenting the source code students often find it very difficult to understand how the algorithm works. As algorithms involve the execution of instructions in a sequential order animations seem to be a suitable visualization tool. Researchers in the area of *algorithm animation* seek to devise methods to create animations that communicate the semantics of algorithms.

The term *algorithm animation* is usually credited to [18]. Many special and general purpose algorithm animation systems have been developed since the video *Sorting out Sorting* (1981) by Roland Baecker, commonly regarded as the first example of algorithm animation, was presented at SIGGRAPH ’81. They range from systems which merely display a movie-like animation to systems which allow complex user interactions such as changing parameters and input data. Prominent examples are Balsa II [17], Zeus [16], XTango [108], Samba [106], Daphnis, JAWAA, GATO, GANIMAL, Jeliot, GASP (II) and VADE, and Leonardo. Numerous other systems exist, many of which were developed in-house at universities and are exclusively used for teaching at these universities. After a strong initial enthusiasm for algorithm animation empirical studies, in many cases, found little or no evidence conclusively supporting the hypothesis that algorithm animation significantly helps students to understand complicated algorithms [62]. Much research has been undertaken in trying to understand how these negative results can be explained, especially as pos-

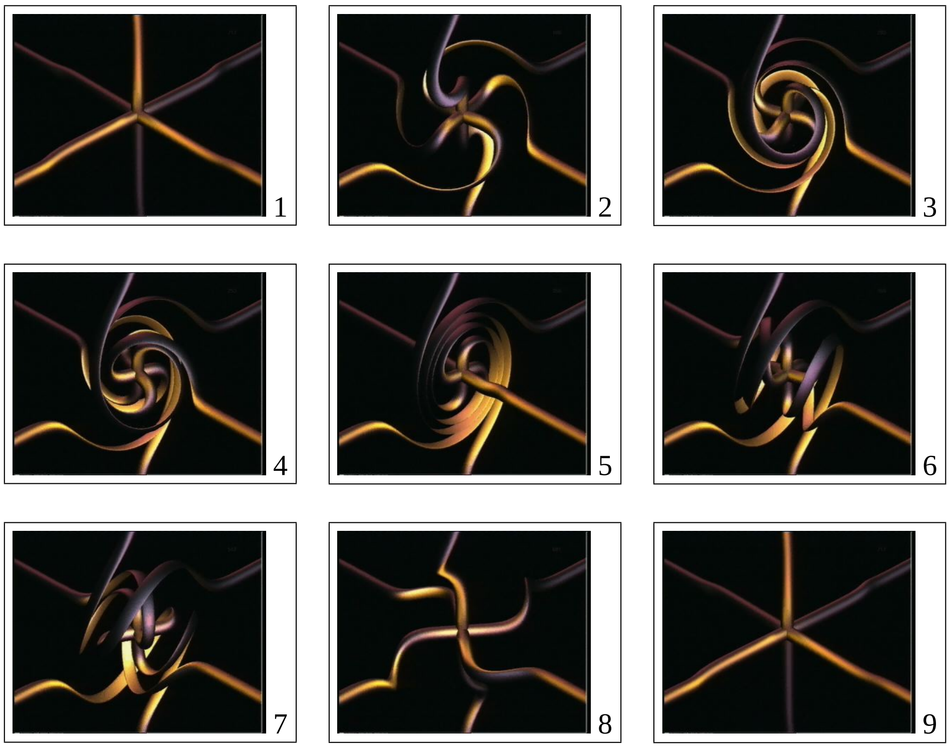


Figure 2.32: This animation is part of an effort to explore possible connections between the spin of fundamental particles and the geometry of space-time. Courtesy of Andrew Norton, School of Mathematics, University of New South Wales

itive findings have also been reported [55, 64, 90]. Many factors which determine the success of animation in teaching algorithms have been identified. Among these are the learning situation in which the animation is presented [62] and the quality of the actual animations themselves [40].

The story and key-frames are usually almost exclusively designed by humans, in most cases by the computer scientists who developed the animation system. These people usually do not have much expertise in animation design and the poor quality of many stories and graphical metaphors is not very surprising. Key-frames are often graphical representations of the state of the machine executing the algorithm and generated semi-automatically by the system according to user-specified metaphors. In many cases, limited choice in graphical representations adds to the poor overall appearance. In-betweens are in most cases computed as linear interpolated movements between positions. This generally does not agree with how humans anticipate movements. Experiments on human perception have shown that humans generally prefer natural motion paths, for example arcs, to movements on a straight line [36, 100, 102].

In some cases linear interpolation of positions is also counter-intuitive with respect to the presented algorithm itself. For example, a linear movement of an arrow along the address-space is an exceptionally bad, but not unheard of, way to visualize a changing pointer. A pointer is set to a new address in constant time by the computer whereas the animation displays a movement which takes linear time in relation to the distance of the two addresses.

The animation methods which we develop in this thesis do not directly address the problem of animating algorithms. Although some approaches might, at least partially, be used in algorithm animation environments, we would expect that animation methods which explicitly consider the semantics of the underlying animated algorithm are more appropriate in such cases.

Graph drawing Another application where animations are used to visualize abstract data is graph drawing. When working with a graph, the user builds a *mental map* of that graph. The phenomenon of "mental maps" is well studied in the areas of geography [53] and psychology [27]. The term *mental map* is commonly credited to Tolman [113]. It refers to the observation that humans tend to build map-like cognitive representations of their environment including metric properties and topological relationships between landmarks. These mental maps are abstract and distorted representations of the underlying real structures. For example, we automatically build a mental map of the city in which we live. The mental map enables us to navigate in the city without carrying a street directory with us all the time. The map contains very detailed information about our local neighborhood and some other important places whereas areas we hardly visit are represented as abstract units. Whole suburbs that we never go to can be represented at the same size as our neighborhood. On the other hand, important transit roads and landmarks tend to be over-emphasized. Figure 2.33 shows the result of an experiment where inhabitants of Westwood were asked to draw how they see Los Angeles [53]. It shows a detailed knowledge of their neighborhood and major transport facilities such as freeways and the international airport. Representations of other parts of Los Angeles are less detailed and more error prone.

Figure 2.34 shows a caricature of how New Yorkers see the United States. Although humorously over-emphasized it shows some principal truth about the New Yorker's point of view. The most important parts are obviously Manhattan and Brooklyn. The Bronx is at about the same level as Boston marking the northern border. To the west things get more and more confused. States change names or move to completely wrong positions. California is split into Hollywood, California, and San Francisco. Florida, as a popular holiday destination, is enlarged and almost borders to New York.

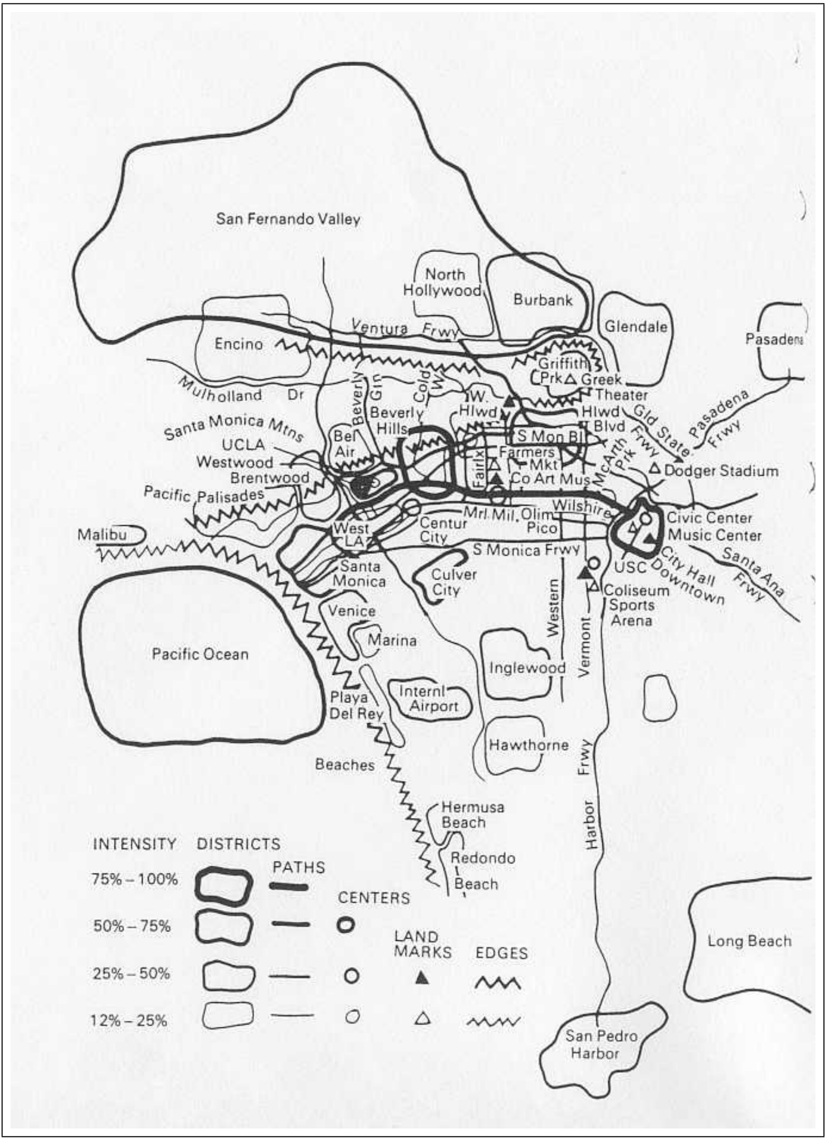


Figure 2.33: Mental Map of Los Angeles of middle-class whites living in Westwood [53]

Although little actual empirical evaluation has been done, it is widely accepted that when working with graph drawings people build a mental map of that graph similar to the mental maps described above [31, 32]. In many applications, however, graphs are not constant but change according to user and program actions. These changes can be visual such as changes of positions or appearance of nodes and edges, or structural such as addition or removal of nodes, edges, or other graphical attributes. When the graph changes beyond a certain threshold the mental map of the user gets destroyed and has to be rebuilt. To increase efficiency in graph drawing applications the number of these events should be minimized. One possible way to achieve this is to try to minimize

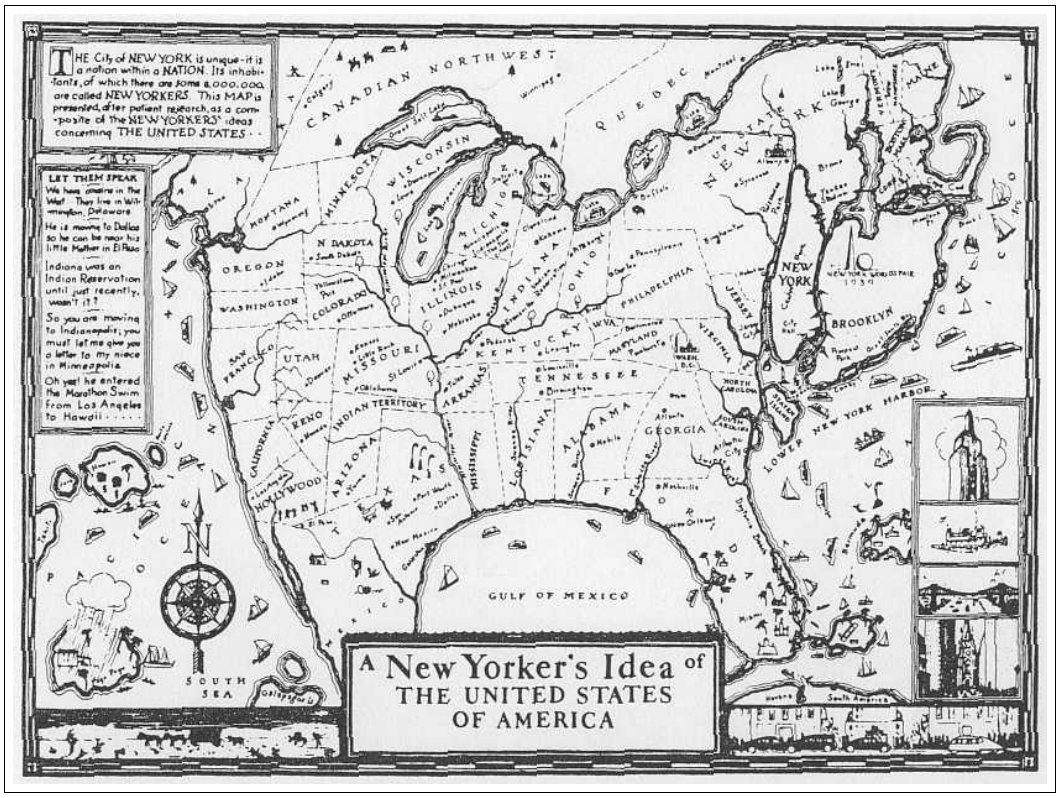

A New Yorker's Idea of THE UNITED STATES OF AMERICA
THE City of NEW YORK is unique-it is a nation within a NATION. Its inhabitants, of which there are some 8,000,000, are called NEW YORKERS. This MAP is presented, after patient research, as a composite of the NEW YORKERS' ideas concerning THE UNITED STATES
CANADIAN NORTHWEST
QUEBEC
GULF OF MEXICO
BROOKLYN
LOWER NEW YORK HARBOR
SOUTH SEA

Figure 2.34: Caricature of how New Yorkers see the United States by Daniel Wallinford. [53]

the changes when changing the graph structure or computing a new layout [13, 31, 32]. However, this is not feasible in all situations.

Another way to help the user maintain the mental map is to communicate the changes in a continuous movement [10, 45]. Several approaches to the inclusion of animations in graph drawing systems have been made, for example in the AGD tool [119] or D-ABDUCTOR [77]. The animations produced by these systems move the nodes and edges of the graph on a straight line from their initial to their final positions. In practice, these animations are often very confusing and seem to be of little help in achieving the task of maintaining the mental map.

In contrast to the general graph drawing system mentioned before, the approach proposed by Yee et.al. in [128] produces animations for the special case of changes between radial layouts of trees. Such changes occur when, for example, a new focus node is selected. By interpolating the polar coordinates instead of the Cartesian coordinates of the node positions they are able to avoid some of the shortcomings of direct linear interpolation. The high quality of the animations is, however, closely linked to the circular nature of the layout and there is no straight forward extension to changes between arbitrarily generated graph drawings.

In graph drawing systems the story of the animation is defined by the changes to the graph. The key-frames are defined by the individual drawings of the graph. As the animations should be created autonomously by the computer, user interaction to specify additional key frames is not desirable and is therefore usually not supported. In all existing systems that we found, in-betweens are computed using a linear path interpolation between initial and final positions of graph elements. In the majority of cases these systems do not produce convincing animations and we found from personal and reported observations, that users often deactivate animations in such systems if this is possible.

The story, as well as the key frames are given by the individual application and therefore cannot be improved by the animation method. The computation of the in-betweens is the only point where improvements in animations can be achieved. In this thesis we therefore focus on developing improved methods and techniques for computing in-betweens in graph drawing environments.

2.3.5 In-betweening

The division of animation sequences into key-frames and in-betweens was introduced to increase efficiency in the production of animation movies in film studios.

To ensure a homogenous appearance of the scenes and characters throughout the movie it would be desirable that all drawings are produced by a small set of artists. To ensure the highest possible quality the best artists should be chosen.

However, many tens of thousands of drawings have to be produced to create a feature-length animation film. Many parallel working artists are therefore needed to be able to create a film in reasonable time. Further, the creative and artistic potential of available artists differs significantly.

As a compromise the senior animation artists only draw certain key frames of the movie, as well as character sheets showing characters in typical poses. The gaps between the key frames are characterized by small changes which can be completed with comparatively low artistic and creative skills by junior artists. Using the key frames and the character sheets as a template high homogeneity as well as quality can be achieved. Due to the low amount of creativity necessary for creating in-betweens computers are a valuable tool to assist artists. In this section we look at how computers are used to create in-betweens in animations. For further reading we refer the reader to [75, 70, pages 44-56].

Point in-betweening

In the following paragraphs we introduce approaches which address the problem of point in-betweening. That is, the problem of computing a motion path for a single point through a given list of key positions.

Linear interpolation The easiest way to move an object from a point P_0 to a point P_1 is to move the object on the straight line connecting P_0 and P_1. If two key frames contain the same number of objects and the objects do not change their shape such a motion can be computed very easily. The following linear function gives the in-between points for the movement of a point from P_0 to P_1:

$$P(t) = tP_0 + (1 - t)P_1, \;\; t \in [0, 1]$$

If an animation sequence is given by several consecutive key-frames continuous interpolation speed can be guaranteed by adjusting the sampling rate of t according to the distances between key points.

In cases where the number of shapes and objects is not constant between key frames, techniques such as described in [19, 70, pages 44-48] and section 2.3.5 can be used to compute a linear movement.

In practice linear interpolations produce bad animations. The main problem with moving objects on a straight line is that the resulting movements do not resemble the typical movement of objects in the real world. To illustrate this problem we look at the following example. Figure 2.35.1 shows key frames of a ball which is thrown from left to right. It is easy for a human to complete the trajectory with the given key frames. A possible result is displayed in figure 2.35.2. Using linear interpolation however, the computed trajectory would look as displayed in figure 2.35.3.

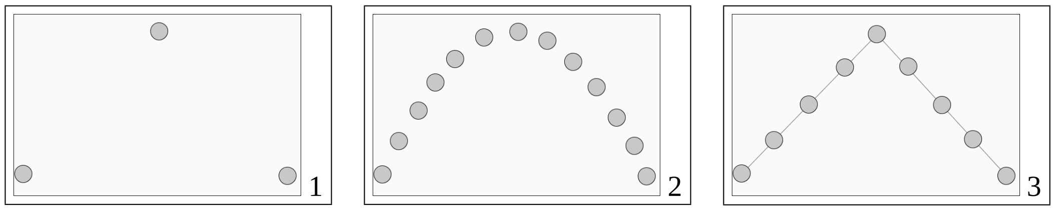


Figure 2.35: Image (1) shows key frames of a ball thrown from left to right. Image(2) shows its natural trajectory as it would be drawn by humans. Image(3) shows the trajectory as it would be computed using linear interpolation of the key frames in (1).

Not only does the ball not follow its natural trajectory, but it also has a very unnatural change of direction at the highest point of its path. The following points can be identified as essential in generating convincing motions:

1. Movements should resemble the natural movements of real objects.

2. Motions should be “smooth” over all key frames.

3. The speed of the movements should be controllable.

The following paragraphs discuss animation methods which generate motion paths which fulfill these conditions. For the purpose of this discussion we restrict animations to movements of points in two and three dimensions. The key frames are specified by the positions and orientations of the points.

In addition to the example from figure 2.35 we use the scenario shown in figure 2.36 for our discussions. It contains three key positions (P_0, P_1, P_2), as well as environmental constraints in the form of lines. A possible interpretation of the scenario could be the path of a person walking along two perpendicular walls. A linear interpolation of the key positions would give an obviously non continuous motion and therefore produce a bad animation.

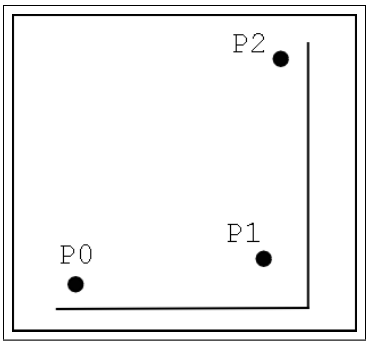


Figure 2.36: Scenario showing a person walking along two perpendicular walls. Key points are specified by P_0, P_1, and P_2

Curves and Splines So far we used the term “smooth” in an informal, intuitive way. For a analytical discussion we have to specify “smooth” in mathematical terms. Although it is clear that the intuitive notion of “smoothness” of a curve is connected to the continuity of its derivatives, it is difficult to determine the precise nature of this relationship. Different opinions exist about the proposition that a function $f \in \mathcal{C}^a$ is smoother than $g \in \mathcal{C}^b$ if $a > b$. That is, that a curve with a higher degree of derivability is smoother than a curve with a lower degree of derivability. A detailed discussion of this topic is beyond the scope of this thesis and can be found in [25, page 56]. For our purposes we assume that for all degrees of derivability larger than or equal to one the perceived smoothness is identical. In fact a higher degree of derivability may lead to unwanted “spikes” in certain key point configurations resulting in unnatural motions. This effect is illustrated in figure 2.37. Image 2.37.1 shows a set of points which is interpolated using a single polynomial. Although the curve is absolutely smooth in the mathematical sense it contains undesirable “spikes”. Image 2.37.2 shows the same point set, this time interpolated using a combination of cubic polynomials. Although the curve is only continuous up to the second derivative it looks more “natural”.

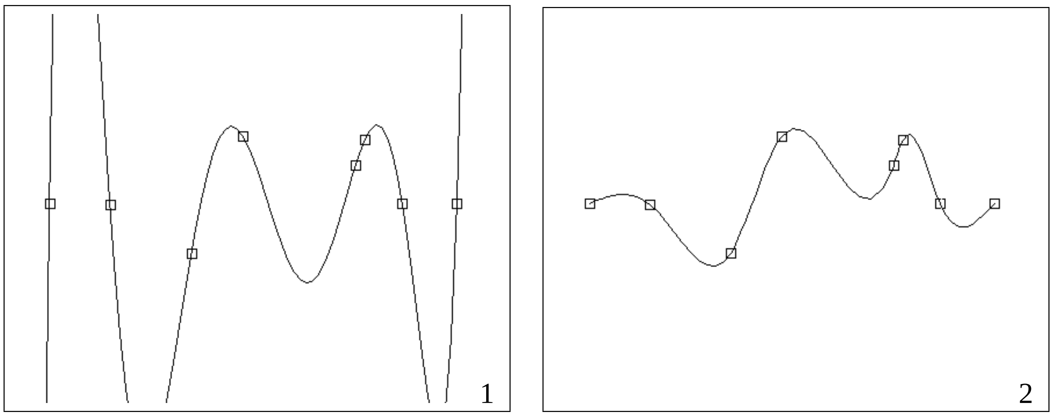


Figure 2.37: Example of smooth point interpolations. Image 1 shows 8 points interpolated using a single polynomial. Although the curve is absolutely smooth in mathematical terms is contains undesirable “spikes”. Image 2, although only continuous up to the second derivative, looks like a more natural interpolation of the point set.

For any set of points S a polynomial f of degree $|S| - 1$ can be found which connects all points of the set in a continuous curve by solving the corresponding set of equations. Although f is element of $\mathcal{C}^\infty$, this form of interpolation has some unwanted properties as we, for example, have seen in figure 2.37. This restricts its use in practice to few, specialized areas such as aerodynamics [120]. In most applications cubic polynomials are used for point interpolation. The curve of a cubic spline is defined by the following set of equations:

$$\begin{aligned} x(t) &= a_1t^3 + b_1t^2 + c_1t + d_1 \\ y(t) &= a_2t^3 + b_2t^2 + c_2t + d_2 \\ z(t) &= a_3t^3 + b_3t^2 + c_3t + d_3, \quad 0 \le t \le 1. \end{aligned} \tag{2.1}$$

.

The curve is uniquely determined by four vectors. This allows the endpoints of the curve segment and the derivative at the end points to be specified in such a way that multiple curve segments can be smoothly combined. A smooth path through an arbitrary number of key positions can thus be constructed.

The coefficients of the equations 2.1 can be extracted into a matrix C. With $T = [t^3, t^2, t, 1]$ the equations 2.1 can thus be rewritten as:

$$Q(t) = [x(t), y(t), z(t)] = TC$$

To avoid having to solve the set of equations for each problem instance, the coefficient matrix can be further decomposed into a basis matrix M which depends solely on which method is used, and a parameter matrix G consisting of the problem instance parameters:

$$Q(t) = TMG$$

Many different methods of parametrization have been developed. Among others Hermite curves [120, pp. 483], Bézier splines [120, pp. 491], Catmull-Rom splines [120, pp. 504], or Akima splines [2]. They differ in the way in which the control vectors are specified. For example, for Hermite splines M is given as

$$M = \begin{pmatrix} 2 & -2 & 1 & 1 \\ -3 & 3 & -2 & -1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{pmatrix}$$

The parameter vector $G = [P_1, P_4, R_1, R_4]$ consists of the start point (P_1) and the end point of the curve segment (P_4), as well as the tangent vectors of the curve at P_1 (R_1)and P_4 (R_4).

Constant animation speed over key points can be guaranteed by using appropriate sampling methods. Figure 2.38 shows a possible path for the scenario from figure 2.36 using a Bézier spline. The path looks much more natural than a path computed using linear interpolation. However, cubic splines have some limitations. In general, they cannot be used to model any arbitrary shape, for example conics [120, page 504], and are not invariant under linear transformations of the control points [120, page 502]. Nonuniform rational B-splines (NURBS) were introduced by Forrest in

1980 [41] to overcome this deficiency. Another problem concerns the additional environmental constraints which may apply to the motion path. Figure 2.38.2 shows another theoretically possible Bézier spline through the same key points. Practically, however, the path is not possible as the person would have to walk through the wall.

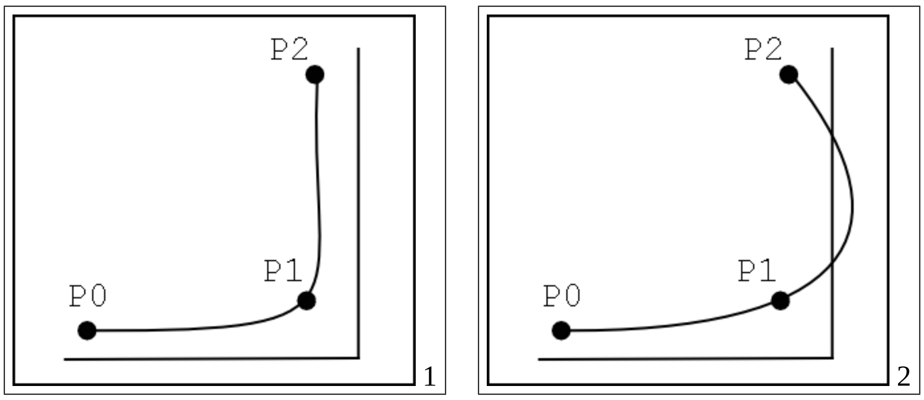


Figure 2.38: Image (1) shows a possible path for the walking person scenario computed using a Bézier spline. Image (2) shows the limits of spline interpolation. Although a valid Bézier spline the computed path is not possible in practice as the person hardly can walk through the wall.

Skeletons Skeletons were introduced by Burtnyk and Wein in 1976 [20]. They try to aid automatic in-between generation by mimicking the way that human animation artists work. For example, when drawing in-betweens for the movement of a human arm the artist does not construct a one to one mapping of each point of the key frames. The artist rather abstracts the arm to a skeleton, constructs the motion phases according to the skeleton and maps the results back to transformations of the drawing of the arm. However, this technique requires a comparatively large amount of human interaction. Figure 2.39 shows examples of skeleton derived images.

In practice skeletons are usually defined as a network of four sided polygons. Points in the drawing get a relative coordinate within the skeleton. After transforming the skeleton the new position of a point can be computed using this relative position.

Quarternions Quarternions are used in three dimensional animations to interpolate between two orientations of a vector. The discussion in this paragraph is mainly based on [25, 120, page 1063]. Further information can be found in [5, 101, 122].

Quarternions were developed by Sir William Rowan Hamilton as a generalization of complex numbers to four dimensions. They have the general form $s + ix + jy + kz$, with $i^2 = j^2 = k^2 =$

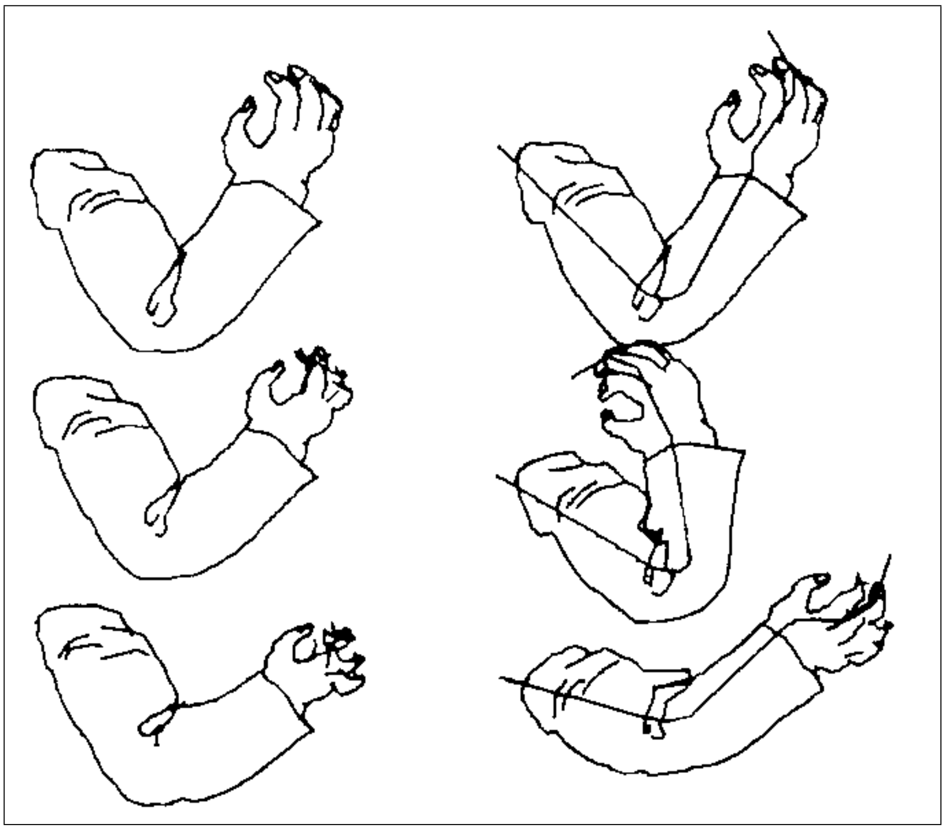


Figure 2.39: Examples of skeleton based drawings [70]

$ijk = -1$ and are usually written as $[s, (x, y, z)]$. The set of Quarternions forms a non-Abelian ring under an appropriate definition of addition and multiplication. Furthermore the set of all possible rotations in three dimensions fits naturally into the set of Quarternions. As the rotation described by a Quarternion is invariant under scalar multiplication the set of all possible rotations is even equivalent to the unit sphere in Quarternion space. The relation of Quarternions and rotations is given as follows:

Theorem 2.3.1 *Let $q \in H_1, q = [\cos\phi, (\sin\phi)n]$, Let $r = (x, y, z) \in \mathbb{R}^3$ and $p = [0, r] \in H$ then $p' = qpq^{-1}$ is p rotated by 2ϕ about the axis n.*

with H the set of Quarternions and H_1 the set of unit Quarternions. A detailed proof of this theorem can be found in [25].

Given two rotations of vectors as Quarternions a continuous interpolation with constant angular speed can be computed by interpolation of the path between the Quarternions on the unit sphere of Quarternion space.

However, there are some problems when using Quarternions to interpolate rotations. Quarternions are based on orientations. Rotations by 0^o and 360^o, although quite different motions, are represented by the same Quarternion. Additional key points can be used to fix this problem. A bigger problem is the fact that there is so far no known way to account for further constraints on movements. When for example interpolating the path of the camera it is desirable that the horizontal orientation of the camera stays constant. It is not possible to model this behavior using Quarternion interpolation.

P-curves So far we assumed that constant speed along a path is the optimal form of animating motion. Although true in many cases there are cases where constant speed can result in unnatural motions. In the example of the flying ball in figure 2.35 constant speed is not appropriate. We would rather like to see an animation where the velocity of the ball decreases on the way to its highest position and increases afterwards. In the example of the walking person (Figure 2.36) we would expect the person to slow down when reaching the bend and possibly accelerating again afterwards.

Modelling objects paths as P-curves was introduced by R. Baecker in 1969 [4]. Using P-curves enables the designer to describe the speed as well as the path of an object. The path of each object is specified by a sequence of points, where each point represents the position of the object at a certain time. As all circles have equal distance in time the velocity of the objects is defined by the spacial distance of the points. Figure 2.40 shows possible P-curves for the examples from figure 2.35 and 2.36.

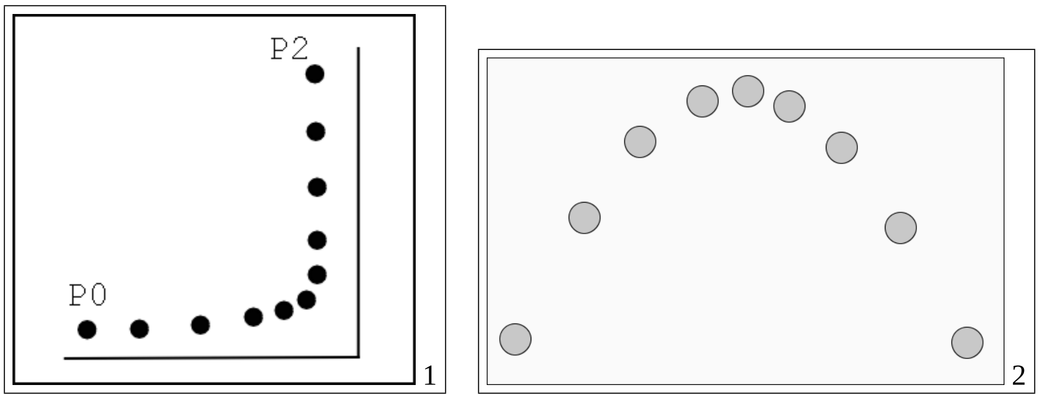


Figure 2.40: Image (1) shows a possible P-curve for the walking person example. Image (2) shows a possible P-curve for the flying ball example.

To increase the smoothness of the path, P-curves can be combined with spline interpolations. The interpolation speed in this case depends on the distance between key points.

Shape and image in-betweening

So far we have discussed techniques to create in-betweens for points and orientations. In some cases, for example when animating a ball, it is possible to represent an object by one point and use the techniques discussed above. Also if we know how to animate an object, for example a walking person, point interpolation techniques can be used to compute the path and the actual motions can be filled in using the known motion sequences. Problems arise however if we want to animate more complex objects. A possible approach in this case is to identify significant points in the object, compute paths for these points and interpolate the intermediate points accordingly. However, care has to be taken when applying this approach. Constraints, such as preserving the shape of the object during the animation have to be guaranteed. Splines for example are in general not invariant to rigid transformations. This might result in unwanted distortions in a shape while performing a simple motion such as rotation.

The terms *morphing*, *metamorphosis*, and *warping* refer to animations where two dimensional shapes and images, or three dimensional objects change their appearance during the animation. The terms are used with some inconsistency in the literature. We use the following definitions in this thesis; metamorphosis is exclusively used for transformations between three dimensional objects; warping is exclusively used for transformations between two images; and morphing is used exclusively for transformations between two dimensional shapes such as polygons or curves.

These techniques, especially warping, are very popular and can be seen in many commercials, music videos, and movies. As we exclusively deal with two dimensional visualizations in this thesis we only discuss warping and morphing in the remainder of this section. An extensive and systematic overview of three dimensional techniques can be found in [66]. The following discussion is mainly based on [126, 127] for warping and [39] for morphing. References to original papers are included in this section.

Warping When warping one image into another certain key features should dominate the transformation. For example, when we warp two faces into each other, the transformation of prominent features such as the eyes, the mouth, or the nose are more important than other parts of the face. It is very difficult for a computer program to identify these features automatically. Although some approaches exist for similar images [49], in most systems the user is required to define the features by hand. To compute a transformation the initial and target image are warped in such a way that the specified key features are mapped onto their equivalents. The actual frame is then computed by interpolating the pixel values of the warped images. Different methods for specifying key features exist:

1. Mesh Warping [125]

   A mesh is super-imposed on both the initial and the target image. The mesh is deformed by hand in such a way that vertices lie at key points of the image and the mesh edges trace the contours of key features. Figure 2.41 shows an example of grid fittings for a given initial and target image.

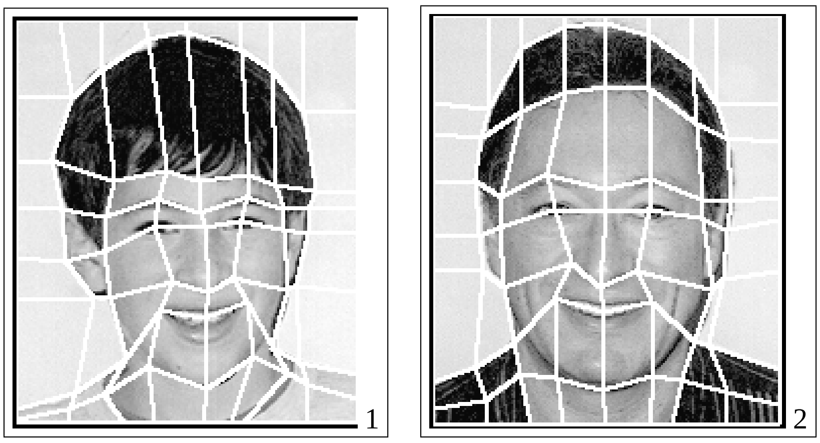


Figure 2.41: Meshes superimposed on the initial and the target image. A warp can be computed by interpolating the meshes. Images courtesy of George Wolberg [125]

   By linear interpolation of the meshes the initial picture can be transformed into the target picture. The pixels in the intermediate pictures are a weighted combination of the initial and the target image pixel values at the interpolated grid positions. To generate smoother transformations, splines can be fitted to the mesh and interpolated instead of the linear mesh segments. This approach is limited to warps which preserve the topology of the image and do not fold.

2. Field Morphing [11]

   Specifying warp meshes is not always easy. Field morphing allows a more efficient specification of the transformation. The transformation between the two images is defined by pairs of user specified corresponding lines in the initial and the target image. Points which do not lie on the line can be interpolated by their distance to the line. If multiple pairs of lines are specified, a weighting according to the distance to a line is used for the interpolation. By marking a few key features such as the eyes or the mouth a warp can be efficiently specified. Field morphing is comparatively slow. As every pixel value depends on all lines, warps with a lot of features become quite expensive to compute. Feature specification also requires skill and experience from the artist as an unfortunate choice of feature lines can produce annoying artifacts.

3. Radial Basis Functions / Thin Plate Splines [92]

   Instead of specifying lines as in the field morphing approach, features are identified by the user in the form of point pairs in the initial and the target image. Two smooth surfaces can be derived from this input. If feature point k has coordinates (u_k, v_k) in the initial image and coordinates (x_k, y_k) in the target image then the surfaces can be interpolated from the points (x_k, y_k, u_k) and (x_k, y_k, v_k). Scattered data interpolation techniques such as thin plate splines or radial basis functions can then be used to define the actual warp function.

4. Multilevel Free-Form Deformation [67]

   A much faster approach is the Multilevel Free-Form Deformation. Feature specification is done using points, polylines and curves. In this approach curves can be efficiently defined by using energy minimizing splines, called *snakes*, which automatically adapt to contours in the image. The warp function is defined using free form deformation manipulation and guided by a hierarchy of control lattices. The derived transformation is continuous and one-to-one.

The correlation of features is usually implicitly specified by the user during the feature definition. If complex structures such as polygons are allowed for feature specification then the problem of finding point correlations and transformations between the shapes arises. Although approaches exist for identifying shape correlations [97], in most systems this has to be done by hand.

Although these methods are very sophisticated and produce stunning results when transforming photo-realistic images into one another, we found that they are not well suited for animating changes in visualizations of relational data. We used three publicly available warping systems to compute example transformations between the two graph drawings shown in figure 2.42. The features were specified by hand identifying structurally important nodes and edges in the initial and target drawing. The results are presented in figures 2.43, 2.44, and 2.45.

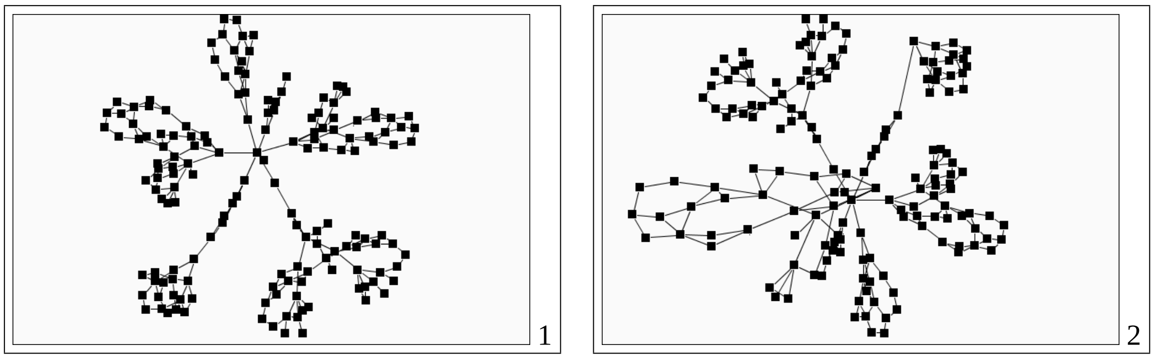


Figure 2.42: To evaluate the use of standard warping methods to relational information visualization publicly available warping systems were used to transform image 1 into image 2

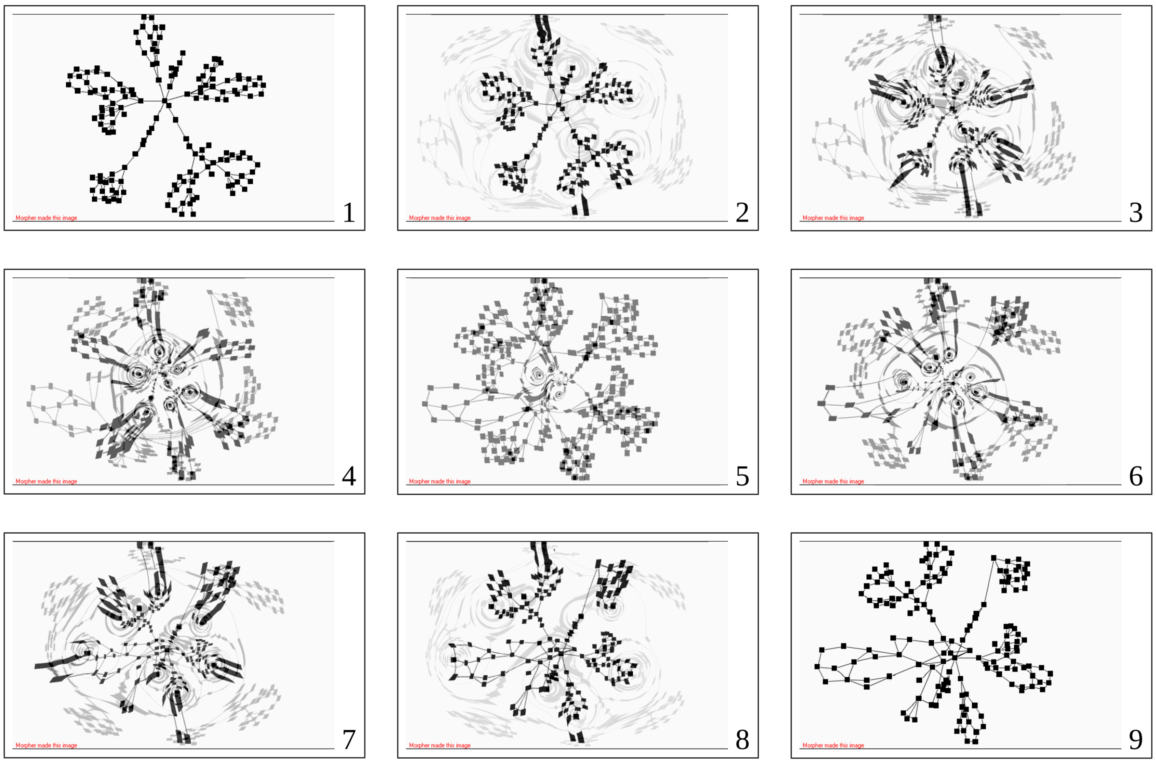


Figure 2.43: Results for Morpher 2000

Some problems become immediately obvious. The animations do not create the effect of moving nodes and edges but rather the effect of one drawing dissolving into the other. This creates good results for photo realistic images. In a graph visualization context however we would like to see the nodes actually move to their destinations. Another major problem is that changes between graph

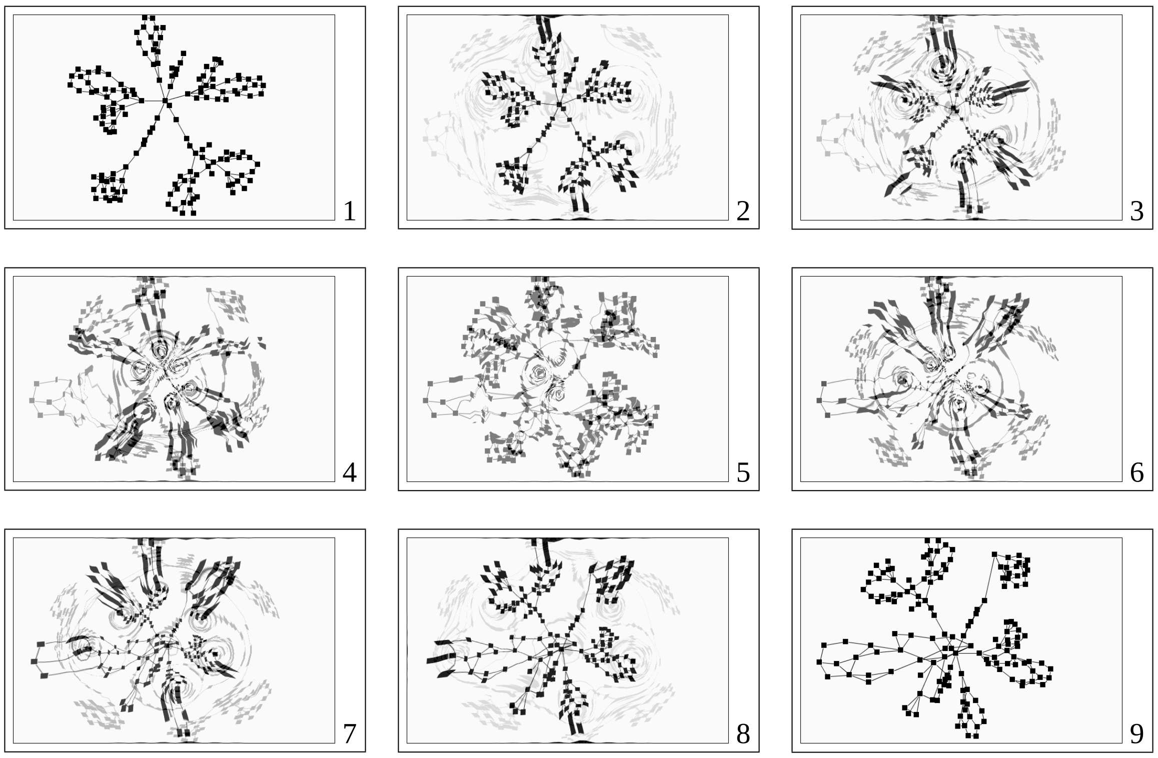


Figure 2.44: Morphman

drawings often imply changes in the layout topology. This fact poses a major problem for warping techniques as most approaches explicitly try to prevent changes in the image topology.

Morphing Several techniques have been developed to transform two dimensional shapes such as polygons into one another. After determining the initial and target position of a vertex in a shape, a motion path between the two locations has to be computed. Apart from the previously discussed option of moving the point on a straight line the following approaches have been developed:

1. The Intrinsic Solution [96]:

   Instead of interpolating the vertex positions of the shape, intrinsic properties such as angles and edge length are interpolated. This technique does not work well with polygons where many vertices lie very close to each other and does not work at all with coincident vertices.

2. The Wavelet Shape Blending Solution [109, 130]:

   The Haar Wavelet Transform can be used to reduce the number of vertices in a polygon. After this transformation intrinsic interpolation can be applied. This approach does not work

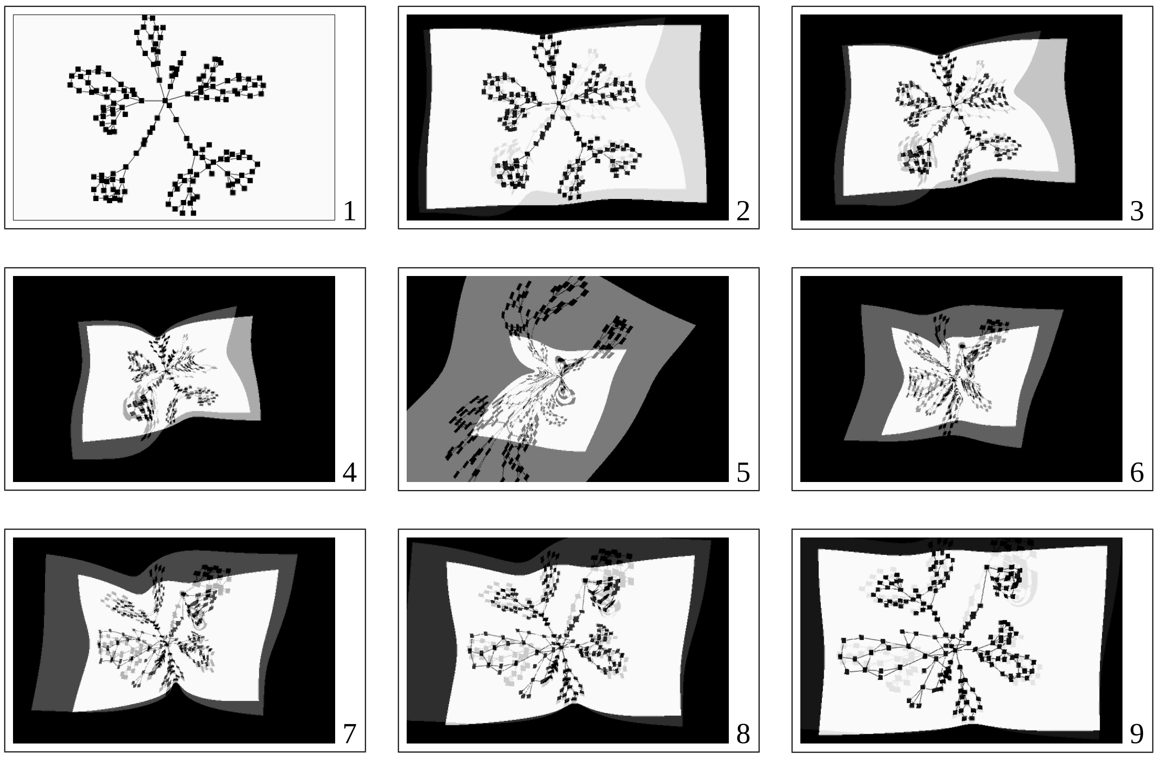


Figure 2.45: WinMorph

well with shapes containing many vertices. It also tends to produce intermediate shapes with broken lines.

3. The Star-Skeleton Blending Solution [99],

   To transform two polygons into one another they are decomposed into star shaped pieces each consisting of vertices of the polygon and a star origin. A star shape is a polygon which contains at least one point, the star origin, from which all points on the boundary of the polygon are visible. After decomposing the polygons we can build a skeleton of the polygon by connecting the star origins with the midpoints of the edges shared by star polygons. A morphing of the polygons can be derived by applying a linear interpolation between the skeletons. This method strongly depends on the initial distribution of vertices.

Again these techniques are not feasible for transforming visualizations of relational data. Apart from the shortcomings mentioned in the description of the techniques, there are additional restrictions of some approaches, for example that the shape must be a closed polygon. This makes morphing techniques unusable for our purposes.

2.4 Visual Perception

Visual perception, especially the perception of motions is a highly complicated task. Major parts of the human brain are dedicated to the processing of visual input, and, so far, computer scientists have not been able to reproduce the human abilities in this area on computers. The human brain is able to derive all information which is necessary to interpret movements in the environment from the changing images projected on the retina of the eyes. Visual motion perception is further complicated by the fact that different kinds of movements can produce identical retinal images. The perceived images when our body moves are the same as when only our eyes move or if the complete visible environment moves. Of course, all of these movements can also happen concurrently. In addition objects can move within the environment, or rather not move in the retinal image if our eyes follow that object, for example flying birds watched through the window of a moving train. To be able to efficiently perceive and correctly understand movements in our environment is a basic necessity for our everyday life. And indeed, we interact successfully with the world most of the time totally unaware of the complex analysis of the retinal images that is necessary to do so.

For an in depth introduction to human perception see [6, 98]. The special case of the perception of moving pictures is addressed in [1, 36, 100].

2.4.1 Apparent Motion

Not all perceived movements are the result of real objects actually moving between places in the real world. When we watch TV we perceive motion of objects on the screen. Of course there are no real objects changing their positions on our TV screen. It is rather the changing images on the screen that induce the illusion of motion. Several forms of apparent motion have been identified [6]:

1. Alpha movement

   Alpha movement is apparent motion resulting from sequentially presented optical illusions. Figure 2.46 shows an example. It displays the famous Müller-Lyer figure. The upper horizontal line is perceived to be shorter than the lower horizontal line, although they actually have the same length. If the two horizontal lines are shown sequentially the horizontal line is perceived to shrink and expand.

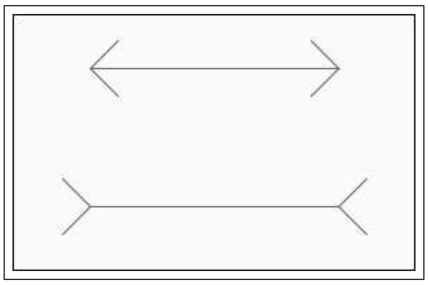


Figure 2.46: The Müller-Lyer figure. The upper horizontal line is perceived to be smaller than the lower horizontal line although they actually have the same length. When displaying the two lines consecutively the line appears to shrink and expand.

2. Beta movement

   Beta movement is perceived when two objects are displayed in succession. Instead of two objects we perceive one moving object. Animations are based on this kind of movement and we exploit this phenomenon for creating animations in the remainder of this thesis. To create the illusion of a motion several technical criteria have to be met. Among others, the targets must be sufficiently close to each other, the time interval has to be right and the targets have to have the right intensity. Korte (see [6]) investigated these factors and found that once the illusion of motion is established it can be maintained even if one of these criteria is altered. Compensating alterations to the other criteria have to be made however. According to Korte distance is proportional to time and intensity whereas intensity is reciprocal to time. It is very easy to fulfill these criteria by imposing some constraints on the frame rate and the speed of objects on the screen. As we never had a case where these constraints were actually violated in practice, we do not investigate this point in more detail in this thesis.

3. Gamma movement

   When the level of illumination is suddenly raised we perceive a radial outward directed motion. We perceive an inward movement when the illumination is lowered. Gamma movement is perceived if the illumination of the whole visual field is changed as well as when it is changed only for specific parts of the visual field.

4. Delta movement

   Delta movement is related to beta movement. When the intensity of the second object is much higher than the intensity of the first object, in addition to the forward movement we finally perceive a backward movement of the object.

Human visual perception is also closely linked to other cognitive processes. Whenever we perceive visual input the brain automatically tries to interpret what it ”sees”. That is, we automatically attribute causality relations or role interpretations to moving objects. Figure 2.47 shows an example. When watching the animation we automatically attribute the movement of the second ball as the result of a collision with the first ball.

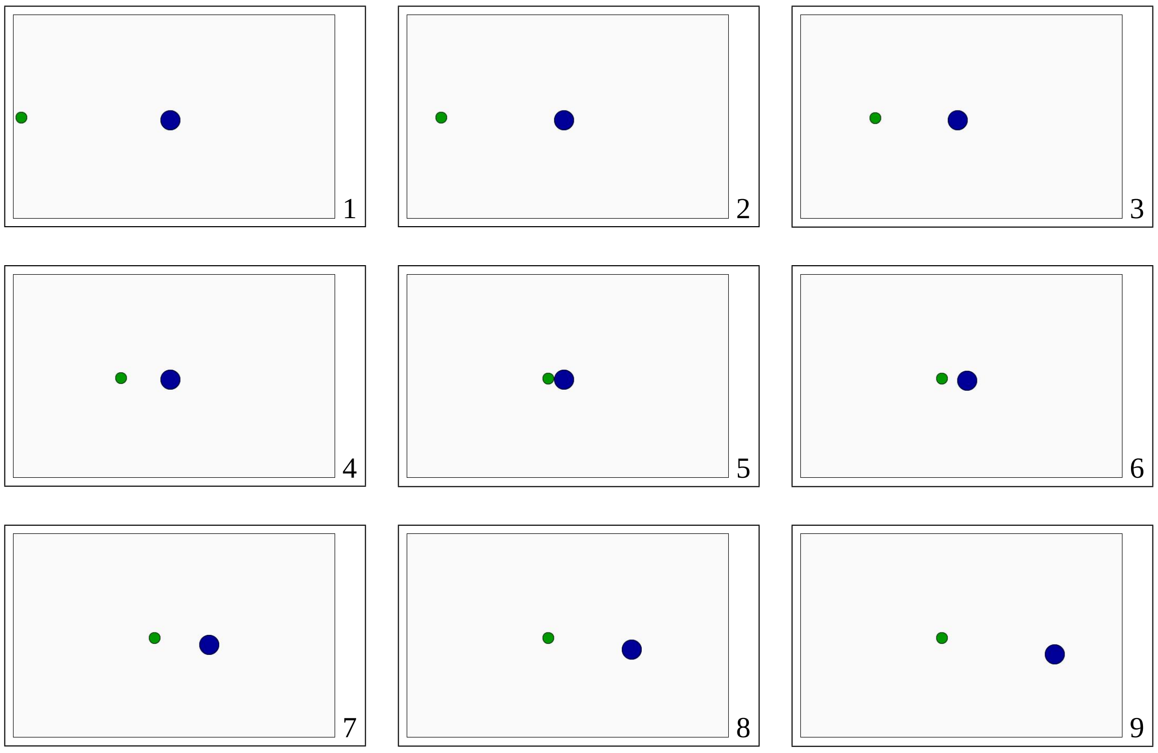


Figure 2.47: When perceiving an animation the brain automatically tried to interpret what it sees. In this case we perceive the movement of the second circle as the result of the collision with the first circle.

The phenomenon shown in figure 2.47 is only one example of a large and complex set of rules and methods (in computer science we would call them heuristics) which the brain developed during evolution. Without these roles it would be impossible to qualify and interpret the large amount of visual information to which we are constantly exposed in the form of retinal images. To identify and understand these mechanisms has been the focus of a long history of research on human perception. Gestalt theory is one of the most important theories about human perception of static and dynamic images and some of the animation techniques introduced in the following chapters are based on principles derived from this theory. We introduce Gestalt theory therefore briefly in the following section. We also relate the results of Gestalt theory to our main application, visualization of relational information. The content of this section is partly drawn from a joint work with Keith

Nesbitt [79].

2.4.2 Gestalt Theory

Perception of visual structure and form, even in stationary images is difficult. A complicated issue is that the overall picture we perceive often seems to extend its component parts. This phenomenon is often described by the saying "the whole is more than the sum of the parts", or in terms of graph drawing, "by looking at individual nodes we do not necessarily learn much about the overall structure of the graph". The Gestalt principles were developed to help explain these holistic characteristics of perception. One of the most appealing properties of these principles is that they are themselves simple to state, understand and apply.

Gestalt Psychology was developed in the 1900s to help counter the associationist view that visual stimuli are perceived as parts and then built into complete images. About 1910, German researchers Max Wertheimer, Wolfgang Köhler, and Kurt Koffka rejected the prevailing models of scientific analysis in psychology and used the principles of field theory to explain cognitive processes which could not previously be explained without a holistic viewpoint [98]. In particular Werthheimer studied the illusion of motion pictures or *apparent motion*.

Gestalt is the German word for form. The Gestaltists proposed a theory of pattern perception that relies on the overall form and is not predictable by only considering individual properties of its components. Factors that impact on the perception of form and impact on how parts are grouped into structural forms are captured in what are called the *Gestalt Principles of Organization*. The importance of these concepts to art and design were explored at the Bauhaus by such artists as Paul Klee, Wassily Kandinsky and Josef Albers.

Our *mental map* of the information underlying a specific drawing of a graph is very dependent on the perceived form or structure of the overall layout. Gestalt principles describe how many elements presented together tend to become grouped into distinct patterns. Hence, these principles strongly influence how the components of a graph drawing, the nodes and links are organized and perceived as a whole.

In the following section we illustrate the Gestalt principles briefly with examples taken from the domain of graph drawing. To demonstrate the significance of the Gestalt principles some examples (Figures 2.49, 2.50, 2.51, 2.53, 2.54) were especially designed in a way that the perceived structure is not equivalent to the real structure of the graph as defined by the edge relations. The following discussion is based on the description of the Gestalt principles as it is given in [52].

The Law of Simplicity

Every stimulus pattern is seen in such a way that the resulting structure is as simple as possible.

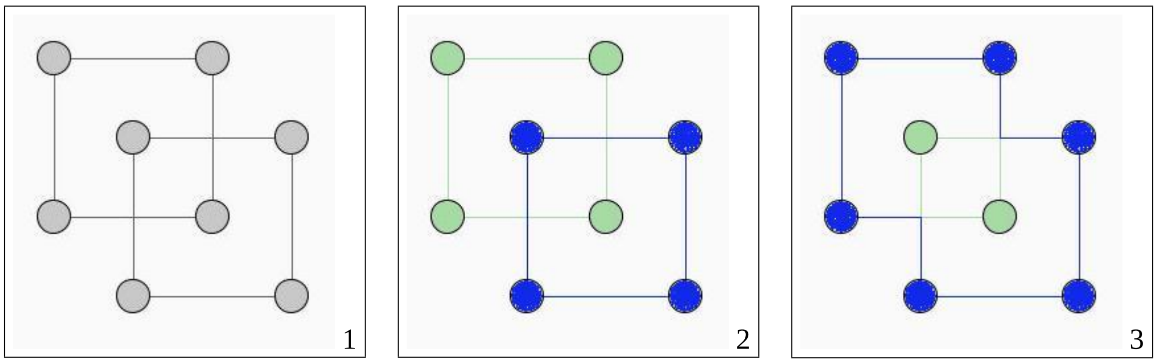


Figure 2.48: Some drawings (1) allow multiple interpretations of the underlying structure (2,3). Most people automatically perceive the more simple structures (2) instead of also possible but more complicated ones (3)

The law of simplicity states that we tend to perceive ambiguous structures (Figure 2.48.1), to be made up of simple shapes (Figure 2.48.2) and reject more complicated, but theoretically also possible, combinations (Figure 2.48.3).

The Law of Familiarity

Things are more likely to form groups if the groups appear familiar or meaningful.

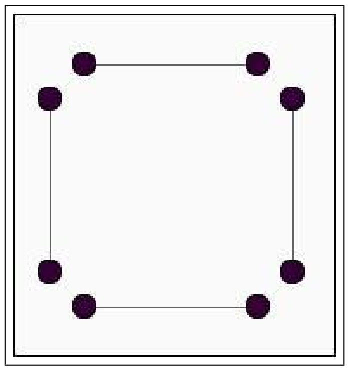


Figure 2.49: The law of familiarity states that we perceive structures that appear familiar such as a rectangle in this figure.

The particular arrangement of the nodes and edges in figure 2.49 leads us to perceive a square. However, there is no such structure in the underlying graph.

The Law of Similarity

Similar things appear to be grouped together

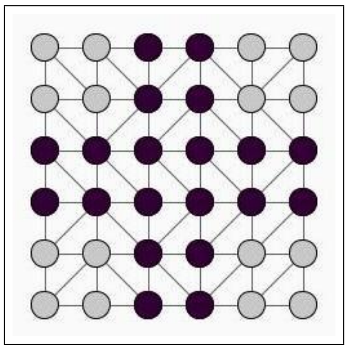


Figure 2.50: The law of similarity states that similar things, such as the black nodes in this graph tend to be grouped together.

When looking at figure 2.50 our brain immediately identifies 5 groups of nodes. The grouping is solely based on the colour and arrangement of the nodes. The edge structure of the graph does not reflect such a clustering.

The Law of Good Continuation

Points that, when connected, result in straight or smoothly curving lines, are seen as belonging together, and the lines tend to be seen in such a way as to follow the smoothest path [52]

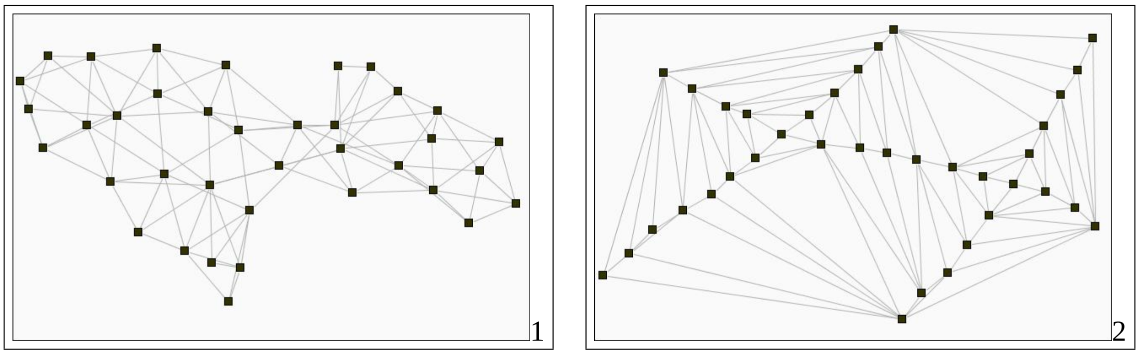


Figure 2.51: The law of good continuation states that we perceive smoothly curving lines and straight lines in smooth paths. In picture (1) we see the drawing of a graph. No special structures are perceived. In picture (2) we see the same graph drawn with different node positions. This time we perceive the nodes to be part of curved lines.

Figure 2.51 shows two possible drawings of the same graph. In image 2.51.2 our brain identifies three strings of nodes. Drawing 2.51.1, however, does not show such structures.

The Law of Connectedness

Things that are physically connected are perceived as a unit. (Figure 2.52).

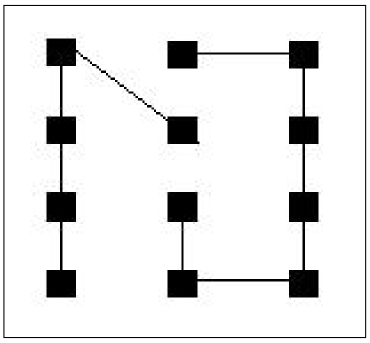


Figure 2.52: The law of connectedness states that we perceive connected objects as a single structure. Hence we see here two connected graph components.

Figure 2.52 shows a grid of nodes. The edges let us easily distinguish two groups of nodes. Most traditional graph drawing approaches are based on this principle.

The Law of Proximity

Things that are near to each other appear to be grouped together (Figure 2.53).

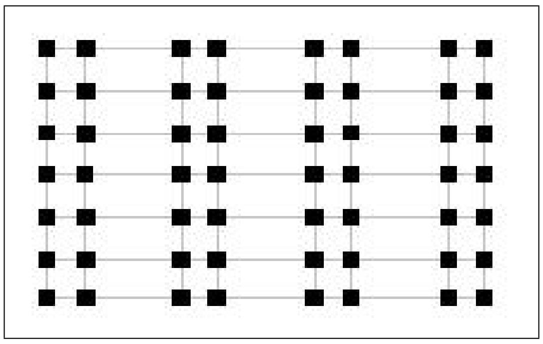


Figure 2.53: This picture displays a grid graph. Due to the special layout of the nodes the law of proximity makes us perceive four columns of nodes instead of a grid.

The law of proximity states that we perceive objects that are close in space to be part of the same structure. In figure 2.53 the spatial arrangement means that our brain perceive four groups of paired columns even though it is theoretically possible to devise other groupings, for example, by row.

The Law of Common Fate

Things that are moving in the same direction appear to be grouped together. (Figure 2.54).

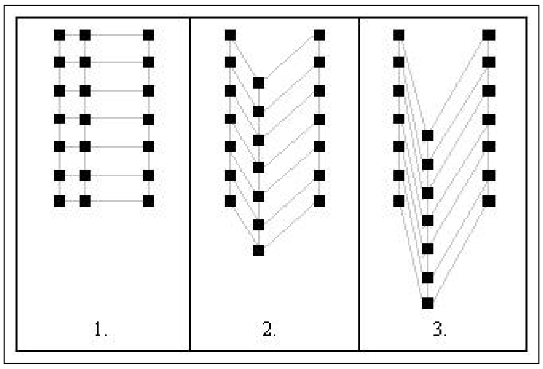


Figure 2.54: The law of common fate states that we perceive objects that move together to be part of the same structure (the second column).

Figure 2.54 shows three different drawings of the same graph. The node column in the middle of the drawing uniformly change their positions between the three drawings. Our brain automatically groups these nodes together. In this thesis we explicitly use this principle in the *Cluster Analysis* animation method in section 6.2 from page 234.

CHAPTER 3

Model

In the main part of this thesis we introduce methods for computing animations in information visualization environments. To be able to do this in a formal and well defined way we introduce some definitions which specify the nature of animation problems in this chapter. Based on these definitions we develop criteria and measures for comparing the visual quality and the efficiency of different animation methods.

Further, we present a general framework for systems providing animations in information visualization contexts. This framework is later restricted to the specific problem domain of relational information visualization which we cover in detail in the main body of this thesis.

Finally in this chapter, we introduce a suit of benchmark tests which we later use to empirically evaluate and compare the quality of the proposed animation methods. The benchmark suit consists of graph visualizations drawn from real world applications, as well as visualizations which are specifically designed to challenge graph animation methods.

3.1 Definitions

Information visualization has many formal definitions. For the purpose of this thesis we use the conventions presented in the following sections. Although they are not derived from any specific existing work, they nevertheless follow the general approach used by most publications in this field.

To display a data item, it is usually associated with a graphical representation. We call this graphical representation a "glyph":

Definition 3.1.1 (Glyph) *A* glyph *is a displayable graphical entity which can be used to represent data. It consists of a two or three dimensional shape definition, as well as graphical attributes such as texture, colour, or transparency.*

Glyphs are the basic building blocks of any “visualization”. To display a glyph, for example on a computer screen, we assign a “location” to this glyph which determines where it is to be displayed. The “location” should also specify the orientation with which the glyph is displayed.

Definition 3.1.2 (Location) *A* location *specifies the position and orientation of a glyph in two or three dimensional space.*

A specific mapping of the elements of a data set to glyphs and locations gives us a well defined way of displaying that data set. In practice, however, most “visualizations” usually contain further graphic attributes. The glyphs are usually displayed in front of a specific background. This can be a simple background colour or a complex scenery. We call this the “environment” of the “visualization”. The “environment” can also contain information about the spectator, for example in the form of camera position and orientation.

Definition 3.1.3 (Visualization) *A* visualization $V = (E, D, G, L, \lambda)$*, consists of an environment specification* E *and a mapping* $\lambda : D \to (G, L)$ *of a set of data items* D *to a set of glyphs* G *and locations* L*.*

Not all elements of the data set have to be represented by a glyph in a visualization. It is possible that several items are represented by one glyph, or that certain items are not represented at all.

During an “animation” we want to smoothly transform one visualization into another. This means that we have to apply continuous functions to the glyphs, locations and environment of the initial visualization which transform them into their equivalents in the target visualization. For our purposes we assume that such a transformation for the given glyphs, locations, and environments always exists. For most display devices, such as computer screens, this is trivially true. For example, simple pixel interpolation would already satisfy continuity for glyph and environment transformations. As the locations are considered to be always specified in a continuous coordinate space, a continuous transformation between the locations of the glyphs also always exists. As the transformations are continuous, we know that there must also exist a distance function for the domain and image of each transformation. That is, we know that there must be a distance function for the glyphs, locations, and environments. This allows us to define a distance function for visualizations which can be smoothly transformed into one another:

Definition 3.1.4 (Distance between Visualizations) *Given two visualizations V_1 and V_2 of a data set D and distance functions between the environments ($\mathcal{E}$), the glyphs ($\mathcal{G}$), and the locations ($\mathcal{L}$) we define the* distance between visualizations *V_1 and V_2 as:*

$$\Delta\left(V_1, V_2\right) = \mathcal{E}\left(E_1, E_2\right) + \sum_{d \in D} \mathcal{G}\left(\pi_1\left(\lambda_1\left(d\right)\right), \pi_1\left(\lambda_2\left(d\right)\right)\right) + \sum_{d \in D} \mathcal{L}\left(\pi_2\left(\lambda_1\left(d\right)\right), \pi_2\left(\lambda_2\left(d\right)\right)\right)$$

with $\pi_x(v)$ being the projection on the x^{th} dimension of vector v.

We deliberately do not exactly specify the individual distance functions between environments, glyphs or locations as they strongly depend on the chosen representations and may vary from application to application. We require however that they are distance functions in the strict mathematical sense, that is $\mathcal{G}$, $\mathcal{L}$, and $\mathcal{E}$ must form metric spaces. Using this distance function Δ, we can define "animation" as:

Definition 3.1.5 (Animation) *An* animation *$\mathcal{A}_\mathcal{D}$ of a data set D is a function from $[0 \ldots 1]$ to the space of visualizations V which assigns a visualization $v \in V$ displaying D to each time $t \in [0 \ldots 1]$. The animation $\mathcal{A}_\mathcal{D}$ is continuous in the interval $[0 \ldots 1]$. That is, for every $\epsilon > 0$ there exists a $\delta > 0$ such that $\Delta\left(\mathcal{A}_\mathcal{D}\left(t_1\right), \mathcal{A}_\mathcal{D}\left(t_2\right)\right) < \epsilon$, for all $|t_1 - t_2| < \delta$.*

Using these definitions we can formulate the problem of computing an information visualization animation:

Information visualization animation problem

Given a data set D and a sequence of visualizations $V_0, V_1, \ldots, V_n$ of D, "efficiently" compute a "good" animation $\mathcal{A}_\mathcal{D}$ with $\mathcal{A}_\mathcal{D}\left(0\right) = V_0$, $\mathcal{A}_\mathcal{D}\left(1\right) = V_n$, and for all i,j with $0 < i < j < n$ there exists t_i and t_j with $0 < t_i < t_j < 1$ such that $\mathcal{A}_\mathcal{D}\left(t_i\right) = V_i$ and $\mathcal{A}_\mathcal{D}\left(t_j\right) = V_j$.

An in depth discussion of the here intuitively used notions of "efficient" and "good" can be found in section 3.2.

In the remainder of this thesis we focus on animation techniques for two dimensional visualizations of relational data as one specific area of information visualization. Although some results from our research can be generalized to other areas, the introduced methods are often optimized to take advantage of properties specific to graph visualization. This restriction allows us to introduce the following, more specific definition of an animation.

Definition 3.1.6 (General 2D graph animation) *A* 2D graph animation *is an animation with the following restrictions. The data set D consists of the nodes and edges of the graph. The set of glyphs consists of simple closed polygons for nodes and poly-lines for edges. Both nodes and edges may have further graphical attributes such as colour, line thickness, fill patterns and others. Locations consist of two dimensional coordinates and one dimensional rotational orientations.*

This is the first work specifically addressing the problem of computing good graph animations. The number of free parameters in the definitions that we have presented so far, however, is too large to cover completely within the scope of one thesis. We have decided to focus on a small set of parameters which we regard to be most critical. Our selection of parameters is based on the following observations:

- The graphical environments in which visualizations are set can have many different forms. It is therefore almost impossible to devise general animation methods for environment transformations. Further, in most cases we can expect the environment to consist of a single background colour which does not change. We therefore do not discuss environment transformations.
- There exists a wide range of warping methods, such as the ones presented in chapter 2.3.5, which can be used to animate transformations between shapes and textures of glyphs. We therefore do not investigate this point further and assume that the shapes of glyphs stay constant between visualizations.
- The problem of identifying point correlations for edges with a changing number of bends between visualizations can be approached using the morphing methods described in chapter 2.3.5 and we therefore restrict our investigations to graph visualizations which do not contain edge bends.

The reduced set of parameters is reflected in the following further restricted formulation of the graph animation problem:

Definition 3.1.7 (Basic graph animation) *A basic 2D graph animation is a 2D graph animation with the following restrictions:*

- *node glyphs are considered to have a fixed orientation and shape.*
- *edge glyphs are straight lines and their location and orientation is implicitly defined by the glyphs of their source and target node.*
- *the only graphical attribute considered is colour.*
- *the environment is empty.*

This definition is used as the basis for all animation methods and measures described in this thesis, and we use the term *graph animation* as a synonym for basic graph animation throughout. Some approaches might contain exceptions, however, and we point them out where appropriate.

So far we have treated animations as a continuous transformation between visualizations. To compute the actual frames of an animation and display them on a computer screen we have to discretise the animation. We do this by sampling the animation function for appropriate values of t. The sampling methods are straightforward and not discussed further in this thesis.

We further assume that for most applications the list of key-frames contains only two entries. This means that we discuss multi-key-frame behavior in the analytical evaluation, but restrict actual experiments to two key frame scenarios.

It is obvious that for any given graph animation problem instance there are an infinite number of valid solutions which vary significantly in the quality of the output and the efficiency with which they can be computed. In the next sections we introduce criteria and measures which define the quality and efficiency of animation methods and thus enable us to compare different approaches.

3.2 Quality Measures for Graph Animations

The overall goal of a graph animation is to help the user to efficiently maintain or adjust the mental map of a changing graph drawing. Major changes to the drawing of a graph usually occur when a user applies a layout algorithm which provides a different view of the graph or when the structure of the graph changes in such a way that makes it necessary to recompute the layout. Examples of structural changes in graphs are collapsing or expanding sub graphs in clustered graphs, navigation in infinite graphs (for example www-based graphs) or graphs encoding dynamically changing data, such as graph A of the 1999 Graph Drawing Contest. Graph A of the 1999 Graph Drawing Contest describes the relationships between the characters of a popular German soap opera. The characters of the soap opera, and especially their relationships, change over time. Figure 3.1 shows part of a poster displaying this graph.

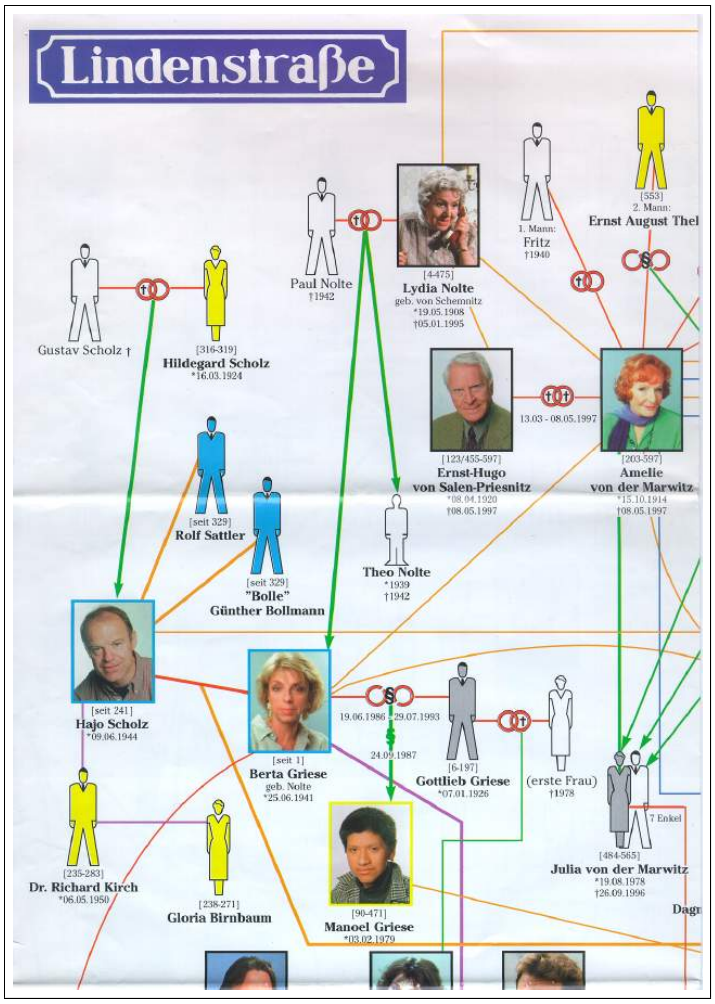


Figure 3.1: Part of a poster showing the characters and relationships within a popular German soap opera

We have identified the list of criteria which we present in the following sections to have a strong influence on achieving the goal of creating an efficient animation. These criteria were derived using theoretical considerations, adopted from existing results in graph drawing [8, 85, 86, 87] and cognitive psychology [36, 100], or are based on empirical data gathered by personal experience and user feedback. Note, that a controlled empirical evaluation and ranking of these criteria by HCI experiments would be necessary to validate these criteria. The lack of precedent research in graph animations, however, implies that we need to explore the solution space first.

Where possible, we derive formal measures for the criteria. As we will see in the following sections, this is not possible in all cases. The discussion of the quality of our animation methods is therefore split into two parts: a strictly formal evaluation where we apply the specified measures as well as an informal evaluation reflecting other criteria.

We can identify three major categories of criteria: technical criteria, static criteria, and dynamic criteria.

3.2.1 Technical Criteria

Technical criteria describe the general conditions which are necessary to compute and present an animation. Generally, the technical framework in which an animation method is embedded has to be able to display approximately 25 frames per second. At 18 frames per second the animation is noticeably jerky. Early movies were filmed and projected at 18 frames per second, resulting in the well known jerkily distorted movements of the actors. At about 10 frames per second the illusion of motion breaks down completely. Two key factors determine the projection speed.

1. The computational cost to compute an animation

   This point is a major concern in all our methods. Our general goal is to provide real time animations to a user interacting with a graph visualization system. This is only possible if the frames can be computed fast enough before or during the animation. Any computation that has to be performed before the animation is displayed bears a high risk of noticeably increasing the response time of the system. Only extremely efficient algorithms can be used here. Although being a very strict and obviously true criteria, it is not well suited as a measure. Whether an animation method can achieve the goal of producing 25 frames per second very much depends on the actual implementation of the animation method and the platform it is executed on. We therefore use a more abstract measure. A method is considered to be superior to another method regarding computation time, if it has a smaller computational time complexity than the other method. A better time complexity does not guarantee a noticeably better performance for a given problem instance, however; both methods could be too slow or fast enough. We can, however, expect the method with better time complexity to scale better for scenarios involving increasing graph sizes and thus be applicable to a wider range of problem instances.

   During the benchmark experiments we measure the actual computing time for both measures.

For the reasons stated above, the recorded times for the animation methods can only meaningfully be interpreted as relative to each other. Two further complications regarding this point have to be mentioned. Firstly, measured time values can be inaccurate. We are able to measure the total time between start and completion of an operation, but we cannot determine how much of this time was spent on the actual algorithm and how much time was spent by the system doing unrelated tasks such as context switches, paging, or garbage collection. In cases where we suspected external influences to have skewed the results of a particular experiment we discarded the results and repeated the experiment. Secondly, in several cases the execution time was close to the maximum resolution of the time measuring method. All results are only correct up to +/-1 millisecond.

We use two measures: First the computational cost for preparing the animation (Init) and second the cost for computing each frame (T/F).

Measure 1 (Animation pre-processing cost (Init)) *The animation pre-processing cost is the complexity class of the operations that are performed after the key frames are specified and before the animation can start. We assume that the number of key frames is a small constant for all considerations.*

We want the user to get the impression that the animation starts immediately after the user input is finished. This means, that the computations which are necessary at this stage should not take longer than the time necessary to display a small constant number of frames. That is, if we assume a linear time complexity for displaying a frame, it seems sensible to require that the preprocessing time for the whole animation does not exceed $O(n \log n)$ for an interactive system.

Measure 2 (Frame computation cost (T/F)) *The frame computation cost (T/F) is the highest complexity class for computing a frame over all frames during an animation.*

If the complexity of computing a frame is significantly higher than the cost of displaying the frame, then we can expect the overall performance of the system to decrease noticeably. Assuming a linear time complexity to display a frame, we therefore require that an animation method has a maximum linear time complexity for computing each frame in order to be considered efficient.

2. The time necessary to display computed image

   The frames not only have to be computed fast enough, they also have to be displayed fast enough by the system. With big graphs and large displays this can become a problem. However, as all our approaches are independent of this step we ignore it in further discussions. We assume that the system which embeds the animation methods is always able to guarantee a playback rate of at least 25 frames per second.

3.2.2 Static Criteria

Each frame of a graph animation displays a drawing of a graph. It seems sensible to deduce that the quality of an animation is influenced by the quality of each drawing. This allows us to use the large body of results which exist on evaluating the quality of static graph drawings [8, 85, 86]. The following criteria seem to have the strongest influence on the quality of a static drawing [87]:

1. Number of edge crossings

   Edge crossings make it difficult to determine the source and target nodes of an edge. A drawing with fewer edge crossings seems to be easier to understand. Edge crossings can be easily counted in a static drawing. We use the following measure to reflect this criteria.

   Measure 3 (Static edge crossings (SEC)) *Given a graph animation A_G of a graph $G = (V, E)$ consisting of n frames $F_0, F_1, \ldots, F_n$ the static edge crossings index of an animation is defined as follows:*

$$sec = \frac{1}{n+1} \sum_{i=0}^{n} \sum_{e_1, e_2 \in E} \sigma_{sec}\left(e_1, e_2, F_i\right)$$

   with

$$\sigma_{sec}\left(e_1, e_2, F\right) = \begin{cases} 1 & \textit{if } e_1 \textit{ and } e_2 \textit{ cross in frame } F, \\ 0 & \textit{otherwise}. \end{cases}$$

   and n being the number of frames in the animation.

2. Direction of flow, Number of bends, Orthogonality

   Although the *direction of flow*, *number of bends*, and *orthogonality* are important criteria for static drawings we decided not to use them in our evaluation. The *direction of flow* is important when users try to trace directed paths in the graph. This is very likely to occur during an animation, and we thus do not consider this criteria. The number of bends is defined by the key drawings and the animation algorithm has little influence on that. On the contrary it would be rather confusing if the initial drawing and target drawing contain many bends and the animation temporarily removes them. Orthogonality requires nodes to lie on grid positions. This can generally not be achieved during a continuous animation; even if we allow isometric transformations of the grid.

3. Display of non-existing structures should be avoided.

   An often neglected problem in graph drawing is the case where the drawing suggests some structure which does not exist in the graph [123]. Figure 3.2 shows an example for two different layouts of the same graph. The second layout could lead the user wrongly to assume that the graph is a simple path. Similar problems can easily occur during an animation, as the human brain tends to be quite imaginative when it tries to interpret moving images [98]. We discuss this point informally where relevant.

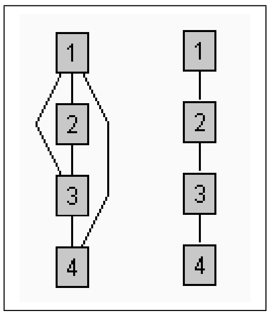


Figure 3.2: Example of a misleading layout

3.2.3 Dynamic Criteria

In addition to the technical and static criteria which we discussed so far, several dynamic, graph animation specific, criteria can be identified.

1. The path of the nodes and edges should be smooth

   Mathematically a function f is considered to be smooth if the derivative of the function is continuous, that is $f \in \mathcal{C}^x$. The vector space $\mathcal{C}^x$ is the space of all functions which can be continuously derived x times. It can be argued whether the intuitive notion of “smoothness” of a function f can be measured by the largest value of x for which $f \in \mathcal{C}^x$ [25, page 56]. For the purpose of this thesis a function f is considered to be smooth if $f \in \mathcal{C}^1$.

Measure 4 (Smoothness) *A graph animation is considered to be smooth if the functions describing the node and edge transformations are in $\mathcal{C}^1$.*

2. Uniform node movement

   If the distance between two nodes in the initial frame is similar to their distance in the target frame, then we would like to see them maintain this distance in all intermediate frames; especially if the two nodes lie comparatively close to each other in the initial and target frames. The optimal distance between two nodes at any given time is considered to be the weighted average of their initial and their target distances.

   Measure 5 (Uniform node movement (UNM)) *Given an animation A_G of a graph $G = (E, V)$ and a distance function $d_{(u,v)} : [0 \ldots 1] \rightarrow \mathbb{R}$ measuring the distance between the positions of two nodes $u \in V$ and $v \in V$ at time t,* uniform node movement *is defined as:*

$$unm = \sum_{u,v \in V} \int_0^1 \left(d_{(u,v)}(t) - \left((1-t)\, d_{(u,v)}(0) + t d_{(u,v)}(1)\right)\right)^2 dt$$

   To evaluate the results of an actual experiment consisting of n frames we use the discrete formulation:

$$unm = \frac{1}{n+1} \sum_{u,v \in V} \sum_{t=0}^{n} \left(d_{(u,v)}\left(\frac{t}{n}\right) - \left(\left(1 - \frac{t}{n}\right) d_{(u,v)}(0) + \frac{t}{n} d_{(u,v)}(1)\right)\right)^2$$

3. Constant edge length

   If the length of an edge in the initial frame is similar to its length in the target frame, then the edge should maintain this length during the whole animation. The optimal length of an edge at any given time is considered to be the weighted average of its initial and target length. This measure is similar to *uniform node movement*, but stresses the distinguished role of nodes which are connected by an edge:

Measure 6 (Constant edge length (CEL)) *Given an animation A_G of a graph $G = (E, V)$ and a distance function $d_{(u,v)} : [0 \dots 1] \rightarrow \mathbb{R}$ measuring the distance between the positions of two nodes $u \in V$ and $v \in V$ at time t,* constant edge length *can be measured as:*

$$cel = \sum_{(u,v) \in E} \int_0^1 \left(d_{(u,v)}(t) - \left((1-t)\, d_{(u,v)}(0) + t d_{(u,v)}(1) \right) \right)^2 dt$$

To evaluate the results of an actual experiment consisting of n frames, we use the discrete formulation:

$$cel = \frac{1}{1+n} \sum_{(u,v) \in E} \sum_{t=0}^{n} \left(d_{(u,v)}\left(\frac{t}{n}\right) - \left(\left(1 - \frac{t}{n}\right) d_{(u,v)}(0) + \frac{t}{n} d_{(u,v)}(1) \right) \right)^2$$

It is obviously true that *constant edge length* is always smaller or equal to *uniform node movement*. Although it is tempting to assume that "UNM $\leq f(G)$ CEL" holds for some graph property $f(G)$, this is generally not true. Figure 3.3 illustrates this. Let us assume a node distance and edge length preserving transformation of figure 3.3.1, for example a rotation by 180^o as shown in figure 3.3.3. Let us further assume that figure 3.3.2 is a frame of a possible animation of this scenario. Clearly, edge length is preserved in this frame and CEL is zero. However, as the distances between the diagonally opposite nodes change, uniform node movement is not given: UNM > 0. No property of the graph can make CEL larger than UNM in this case.

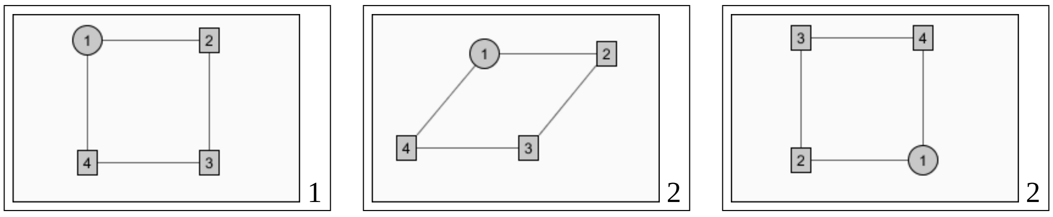


Figure 3.3: Example illustrating that generally "UNM $\leq f(G)$ CEL" does not hold.

4. Minimize temporary edge crossings.

Edge crossings in a graph drawing generally reduce readability. This is also valid for animations. If the animation avoids introducing unnecessary edge crossings on the way, then it is easier for the user to follow the movements. If two edges which did not cross before suddenly cross each other, then this imposes an additional cognitive load on the user. In this measure we count changes of edge crossings. As it is a discrete counting, no continuous formulation of this measure exists.

Measure 7 (Temporary edge crossings (TEC)) *For a given graph animation A_G of graph $G = (E, V)$,* temporary edge crossings *is defined as:*

$$tec = \lim_{n\to\infty} \sum_{t=1}^{n} \sum_{e_1,e_2 \in E} \sigma_{tec}(e_1, e_2, t)$$

with

$$\sigma_{tec}(t, e_1, e_2) = \begin{cases} 1 & \textit{if } e_1 \textit{ and } e_2 \textit{ do not cross in frame t-1, but cross in frame t}, \\ 0 & \textit{otherwise}. \end{cases}$$

and n being the number of frames in the animation.

To measure the results of an actual experiment consisting of a fixed, finitie number of n frames, we use:

$$tec = \sum_{t=1}^{n-1} \sum_{e_1,e_2 \in E} \sigma_{tec}(e_1, e_2, t)$$

5. Minimize node path length

   To decrease the response time of the system, the animation should be as short as possible. If the maximum path of all nodes is small, then the animation can be performed more precisely in the given time and thus is easier to follow.

Measure 8 (Node path length (NPL)) *For a given graph animation A_G of a graph $G = (E, V)$ and functions $x_v(t)$ and $y_v(t)$ giving the x and y coordinates of node v at time t,* node path length *is defined as:*

$$npl = \sum_{v \in V} \int_0^1 \sqrt{(x_v'(t))^2 + (y_v'(t))^2} dt$$

We assume that $x_v(t)$ and $y_v(t)$ are at least piece-wise differentiable. In the case that the functions are not totally differentiable we simply add up the integrals over the pieces. When evaluating results of actual experiments consisting of n frames, we use the discrete formulation:

$$npl = \sum_{v \in V} \sum_{t=1}^{n} \sqrt{\left(x_v\left(\frac{t-1}{n}\right) - x_v\left(\frac{t}{n}\right)\right)^2 + \left(y_v\left(\frac{t-1}{n}\right) - y_v\left(\frac{t}{n}\right)\right)^2}$$

6. Maintain a minimum distance between nodes which do not move uniformly.

If nodes lie close to each other, then it is more difficult to follow their individual movements than if they are further apart. However, if two nodes lie next to each other in the initial drawing and in the target drawing, it would be better to move them uniformly and lying close to each other to their destination, than to separate them first. We also found that if two nodes intersect only briefly, then it is easier to follow their movements than if they intersect for a longer time. As we have no clear empirical evidence to define an optimal minimal distance we only take into consideration whether nodes do actually overlap or not. We derive the following measure counting changes in node intersections:

Measure 9 (Unnecessary node intersections (UNI)) *Given a graph animation A_G of a graph $G = (V, E)$,* unnecessary node intersections *is defined as:*

$$uni = \sum_{v_1, v_2 \in V} \int_0^1 \sigma_{uni}\left(v_1, v_2, t\right) dt$$

with

$$\sigma_{uni}\left(v_1, v_2, t\right) = \begin{cases} 1 & \text{if } v_1 \text{ and } v_2 \text{ intersect at time } t, \\ 0 & \text{otherwise}. \end{cases}$$

To evaluate the results of an actual experiment consisting of a fixed number of n frames, we use:

$$uni = \frac{1}{n} \sum_{v_1, v_2 \in V} \sum_{t=1}^{n-1} \sigma_{uni}\left(v_1, v_2, \frac{t}{n}\right)$$

7. Maximize symmetry

Symmetry in a graph drawing helps the user to understand the structure of a graph. In an animation symmetry of movement makes it easier to understand the structure of the movement. A formal measure for symmetric node movement can probably be derived by extending the model in [72], but is not without problems. Especially symmetric motions which cannot be directly derived from static symmetries of the individual frames, so they exist, would have to be investigated in detail. Due to the computational complexity compared to the small numbers of cases showing symmetries we did not implement a formal measure for this case. Where appropriate, we give an informal discussion on expected behavior of a method regarding symmetry.

8. Maximize use of automated cognitive sub-systems

   As we have seen in the discussion of perceptional mechanisms in section 2.4, the human brain has a predisposition for certain ways of perceiving visual information. Complex analysis and structuring of the visual input is performed by automated, independent cognitive subsystems. We can expect the amount of communicatable information to increase if part of the cognitive load can be shifted from general problem solving areas of the brain to automated sections. For example, humans are very good at recognizing and interpreting human facial expressions. Chernoff tried to exploit this fact in his work on encoding information as facial expressions [21].

   On the other hand we can expect the cognitive load to increase if the animation is counter-intuitive. For example, if an object is translated and rotated at the same time, experiments have shown [36, 100] that we expect the object to perform the translation on an arc instead of a straight line. It is difficult to derive a general mathematical formulation of these mechanisms, although it is possible for some special cases. For example, our brain is highly specialized for perceiving and processing movements of three dimensional rigid objects in space. Projected on the plane, these transformations are identical to the set of affine linear transformations in $\mathbb{R}^2$. We exploit this feature in the *Linear Regression Analysis* animation method in section 6.1.1.

   As no general formulation of how to measure the utilization of cognitive sub-systems is known, we discuss this point informally where appropriate.

Some of the metrics which we presented in the previous sections and combinations of some of these metrics are NP-hard to optimize. An extensive overview of the computational complexity of the static metrics can be found in [8, section A.6, pp. 354-358]. The dynamic metrics are not so easily approachable by the traditional methods of computational complexity. We investigate the following problem in detail: Given a scenario, we want to answer the question whether it is possible to transform the initial drawing into the target drawing with the optimum value of zero for *constant edge length* and *temporary edge crossings*. We can show that this involves a well-known NP-hard problem, called the *Ruler Folding Problem* [124]. The *Ruler Folding Problem* is defined as follows:

Definition 3.2.1 (String graph) *A string-graph is a connected graph of degree 2.*

Definition 3.2.2 (Ruler Folding Problem) *Given a string-graph with n edges where each edge has a fixed length and the edges are allowed to rotate freely around their nodes, can the string graph be folded in such a way that it occupies a line segment of length at most k?*

Consider the scenario in figure 3.4. The graph underlying the scenario consists of two con-

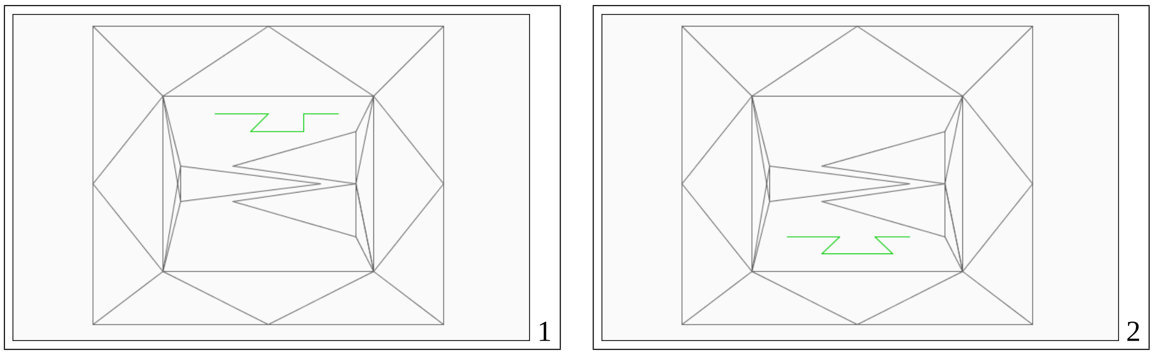


Figure 3.4: Scenario illustrating that optimizing *constant edge length* and *temporary edge crossings* at the same time is generally NP-hard. Moving the green subgraph into the lower section of the enclosed space requires it to fold onto a line segment of fixed size. Deciding whether this is possible is known to be NP-hard [124].

nected components. We refer to the (grey) outer component as the *framework* and to the (green) inner component as the *ruler*[1]. The faces of the *framework* are triangles and there exists a path from the interior of each triangle to the interior of each other triangle through adjacent triangles of the *framework*. The resulting structure is therefore rigid. This means that we cannot deform the *framework* in any way that changes angles or distances between the nodes without increasing the values for *constant edge length*. The *ruler* consists of a string graph and has to move from the upper section into the lower section of the space enclosed by the *framework*. To do this it has to pass a “barrier” which is formed by three interlocked triangles of the *framework*. The initial drawing with the associated roles is displayed in figure 3.5. The barrier-triangles lie arbitrarily close to each other and thus do not allow any significant deformation or rotation of the *ruler* while it is in the barrier

[1]See also figure 3.5.

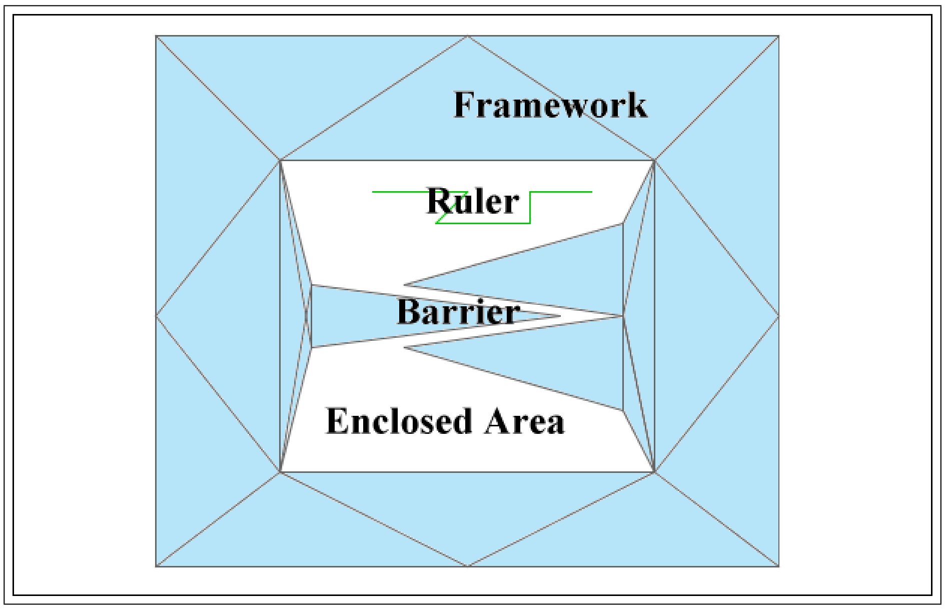


Figure 3.5: Image 1 from the scenario displayed in figure 3.4. The parts important for this scenario have been labelled appropriately.

section. Let us further assume that the enclosed area before and after the barrier is sufficiently large to easily allow all necessary folding operations without any temporary edge crossings. Let k be the distance between the rightmost end point of the left barrier triangle and the base of the other two barrier triangles. In order to move the *ruler* to its target position without changing the length of any edge and without introducing any edge crossing we have to fold it onto a line segment of size at most k. To decide whether this is possible is equivalent to the Ruler Folding Problem which is known to be NP-hard [124]. We can thus state:

Theorem 3.2.1 *Minimizing* constant edge length *and* temporary edge crossings *during a graph animation is NP-hard.*

It is easy to see that this theorem also holds for connected graphs. We can modify figure 3.4 by connecting the *ruler* to the endpoint of the left barrier-triangle with an additional string graph. This additional string graph can consist of an arbitrary number of arbitrary small line segments, thus not interfering while the ruler moves through the barrier section.

In many cases not all criteria can be optimized in one solution. For example, it might be necessary to move nodes along a non-optimal path to avoid edge crossings, or it might be preferable to accept some edge crossings instead of watching the graph untangle in a complex and confusing way. Thorough HCI experiments would be necessary to apply a ranking or even weighting to these criteria. However, these are beyond the scope of this thesis. We therefore treat the measures independently and give an informal discussion on the observed importance of individual measures where appropriate.

To be able to evaluate and compare the different animation methods which we introduce in this thesis, we apply each animation method to a suit of benchmark tests which we introduce in section 3.4. During the benchmark tests we record the measures which we have described in this section. To which degree the recorded measures approximate the continuous measures depends on the quantization of the animation function. To ensure that the recorded measures can be meaningfully compared we require that all animations have the same length, that is 101 frames in our case. It can be argued that for more complex animation scenarios, longer animation times could help the user in understanding what is happening on the screen. Although this is probably true for HCI experiments, it does not affect the purely numerical experiments which we conduct in this thesis. To ensure that the measured results are comparable between different scenarios and animation methods, we further require that the regulator function which is used to control the speed during the animation is the identity function.

3.3 Framework for Graph Animation Methods

A graph animation method has to compute an animation according to a given list of visualizations of that graph. We assume that the method has full access to all necessary information, that is, node and edge correlation between visualizations, as well as access to position and glyph information. We can identify the following tasks that a graph animation method has to perform:

1. Analyze the list of visualizations and devise an animation strategy.

2. Repeatedly compute frames between two key frames.

We assume that the embedding graph visualization system is able to display the frames once they are computed. We look at these steps in detail in the following sections.

3.3.1 Analyze Input and Devise an Animation Strategy

The input for a graph animation method consists of a list of graph visualizations. They define the key frames of the animation. The animation method has to analyze these visualizations and devise an animation strategy. As this task has to be completed before the first frame is shown, it is essential that it can be performed very efficiently in interactive systems.

3.3.2 Compute Animation Frames

During the animation between two key frames we expect the nodes and edges to change their positions and move to their destinations. However, this is not possible for all nodes and edges. If, for example, a node exists in the initial frame, but does not in the target frame, then no motion for this node can be specified. We might, however, want to animate the vanishing of this node. To just leave the node at its initial position and let it fade away while the other nodes move to their destinations is generally not a good solution. If the other nodes of the graph perform a very structured movement, such as a rotation, then the vanishing nodes can destroy this structure by unnaturally staying at their initial positions. This observation leads to the following three independent parts of an animation.

Compute pre-motion frames During the pre-motion stage all animations should be executed which cannot sensibly be done during the motion phase. Examples are fade outs for node and edges which do no longer exist in the target key frame.

Compute motion frames During the motion phase the nodes and edges move between the key frame positions. To display the animation on a computer screen, it is necessary to discretise the animation function. The animation method has to be able to provide the appropriate frames for the resulting discrete values of time $t \in [0 \ldots 1]$. Two approaches are possible:

1. The animation method provides a function which is able to compute a frame for an arbitrary value of t. The speed of the movement can be controlled by modifying t using an appropriate regulator function. Examples of commonly used speed functions are shown in figure 2.30 on page 39. We use this approach for most of the proposed methods.

2. Not all animations methods are able to provide such a function as we will see in section 5.1 on page 146. We therefore alternatively allow an animation method to iteratively produce a previously fixed number of frames. The animation method has to provide a function with which the number of frames for the animation can be set. Animation methods which implement this approach are usually not able to adjust the speed of the animation once it has started.

Additional transformations such as changes in colour or shape can also take place during the motion phase.

Compute post-motion frames Similar to the pre-motion phase, we allow a post-motion phase to perform final animations which do not make sense during the motion phase. Examples are fade ins of nodes and edges which did not exist in the initial key frame but do exist in the target key frame.

3.4 Benchmark Suit

In addition to the analytical evaluation of each animation method, we also conduct experiments to gather empirical data. The scenarios on which the experiments are based were designed to reflect real world situations, as well as especially challenging artificial situations. We introduce these scenarios in the following sections.

3.4.1 Scenario 1

Scenario 1 is based on a graph taken from a dataflow analysis tool for telecommunication services. It contains 158 nodes and 232 edges. The node labels have been removed on request by the owner. The initial and the target drawing are shown in figure 3.6. The initial drawing, shown in figure 3.6.1 was created using a Sugiyama style layout algorithm. The drawing displays the hierarchical structure of the data. Hierarchical approaches such as Sugiyama style algorithms usually add bends to edges to reduce edge crossings in the drawing. As we only consider graphs with straight edges, all bends have been removed from the scenario. The target drawing, shown in figure 3.6.2 was created using a spring embedder algorithm. The second drawing displays proximity information about the data by positioning those nodes close to each other which are connected by an edge.

3.4.2 Scenario 1b

Scenario 1b uses the same drawings as scenario 1. This time however the role of the initial and the target drawing are reversed.

3.4.3 Scenario 2

Scenario 2 shows a 180^o degree rotation of a graph. The initial and the target drawings are shown in figure 3.8. The graph is taken from the demo-graph selection of the Metis graph clustering tool. It

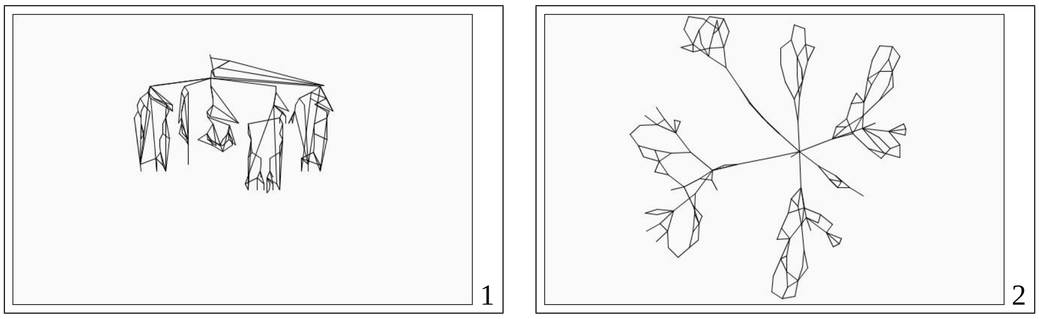


Figure 3.6: Scenario 1: A dataflow graph. Image 1 shows the initial drawing which was generated using a Sugiyama style algorithm. Image 2 shows the same graph drawn using a spring embedder algorithm.

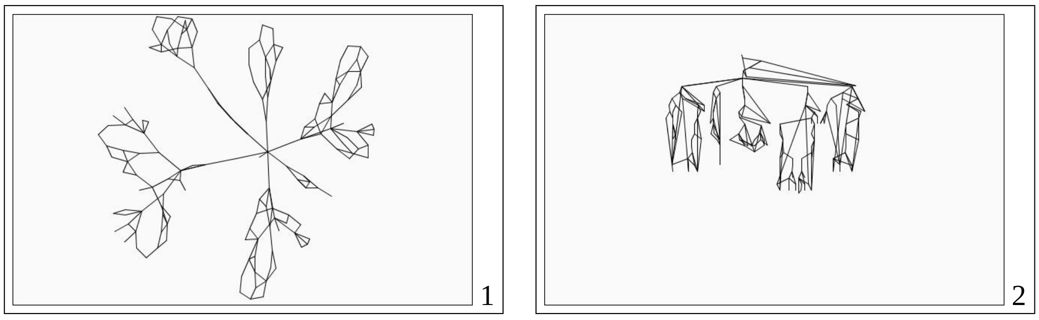


Figure 3.7: Scenario 1b: A dataflow graph. Image 1 shows the initial drawing which was generated using a spring embedder algorithm. Image 2 shows the same graph drawn using a algorithm Sugiyama style algorithm.

contains 1036 nodes and 1868 edges. Rotations by large angles seem to pose significant problems for naive animation techniques. For example, moving the nodes on a straight line from their initial positions to their target positions would result in the graph collapsing to a single point during the animation.

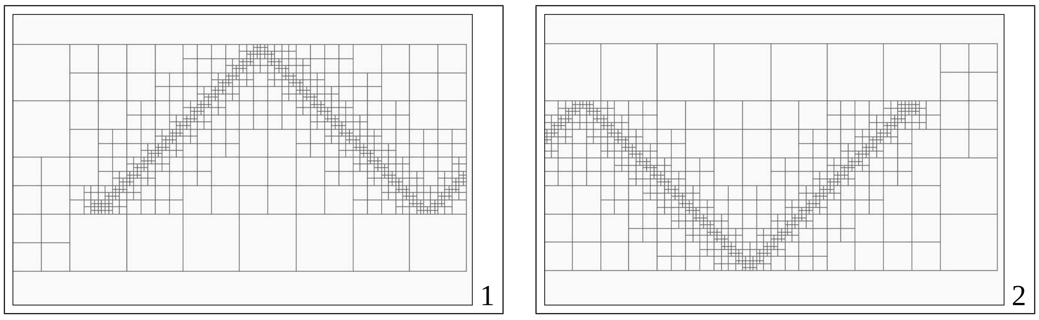


Figure 3.8: Scenario 2: The initial and the target drawing show the rotation of a graph by 180^o.

3.4.4 Scenario 2b

Scenario 2b shows a 180^o degree rotation of the same graph which was used in scenario 2. In addition to the rotation, all nodes were displaced by a small random offset after the rotation. The scenario is shown in figure 3.9. This scenario challenges methods which try to analyze the initial and target drawing to find structured movements.

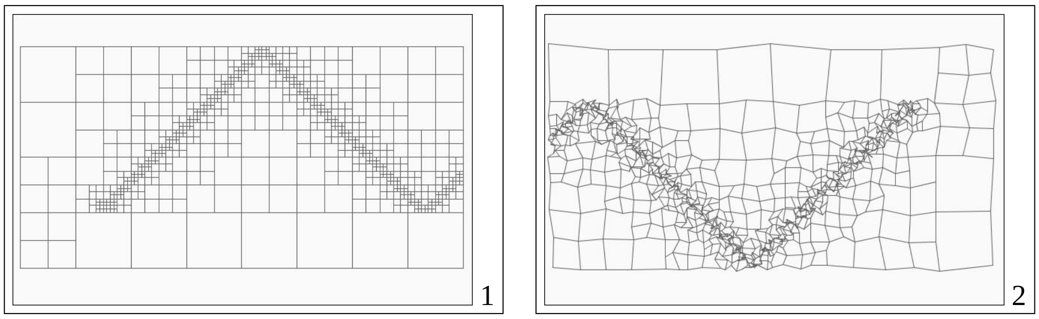


Figure 3.9: Scenario 2b: The initial and the target drawing show the rotation of a graph by 180^o. In addition all nodes were displaced by a random offset after the rotation.

3.4.5 Scenario 3

Scenario 3 shows a scenario where sub-graphs move in different ways. The graph is a tree and it is drawn using a traditional tree layout algorithm in figure 3.10.1. To be able to have a closer look on a particular sub-tree, the sub-tree is drawn twice as big in figure 3.10.2. To make room for the bigger sub-tree, another sub-tree is rotated to the top. The graph contains 177 nodes and 176 edges.

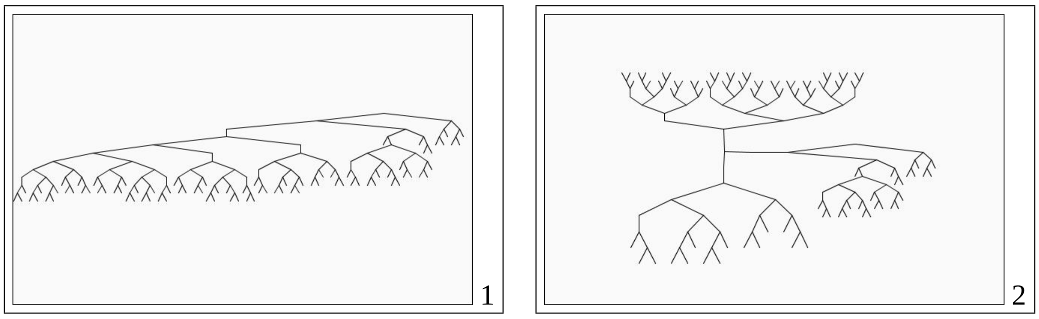


Figure 3.10: Scenario 3: This scenario consists of two drawings of a tree. To be able to have a closer look on a particular sub-tree of image 1, the sub-tree is drawn twice as big in image 2. To make room for the bigger sub-tree, another sub-tree is rotated to the top

3.4.6 Scenario 4

Scenario 4 shows a communication network. The nodes of the graph represent telephones and two nodes are connected if there has been a call between these phones. These kinds of graphs are used in criminal investigations. The was taken from category B of the 1996 graph drawing competition of the International Symposium on Graph Drawing. As the original initial drawing contained edges with bends, the bends have been replaced by dummy nodes in this scenario. The target drawing was created by applying a spring embedder algorithm to the graph. The graph contains 120 nodes and 202 edges.

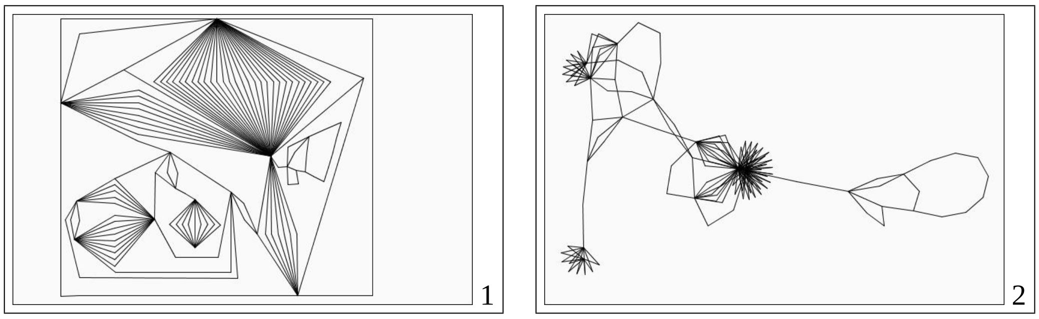


Figure 3.11: Scenario 4: Different drawings of a communication network. Nodes represent telephone numbers. Two nodes are connected if there has been a call between these two numbers.

3.4.7 Scenario 5

Scenario 5 also shows data from a telecommunication network. The nodes of the graph represent telephone numbers and two nodes are connected if there has been a call between the numbers. The graph was taken from category C of the 1997 graph drawing competition and contains 345 nodes and 626 edges. The initial drawing and the target drawings were computed by using a spring embedder algorithm. Some clusters of nodes are collapsed in the initial and the target drawing and are thus only visible in one drawing. The scenario is shown in figure 3.12. Image 3.12.1 shows the initial drawing and figure 3.12.4 shows the target drawing. In image 3.12.2 the nodes which exist in the initial drawing but are hidden in the target drawing are painted grey. In image 3.12.3 the nodes which exist in the target drawing but are hidden in the initial drawing are painted grey.

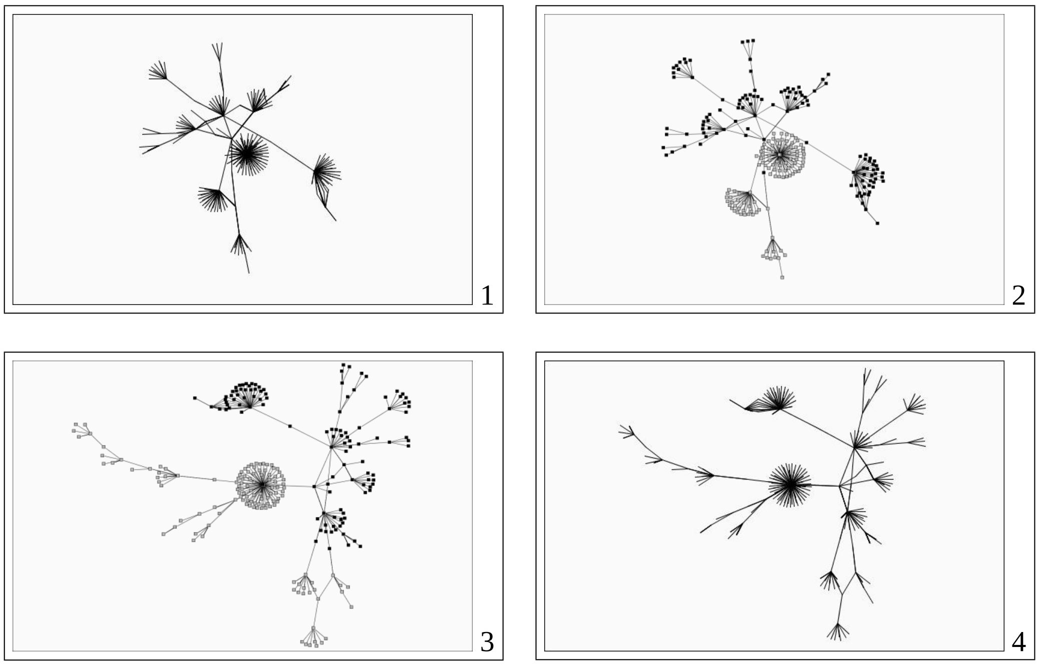


Figure 3.12: Scenario 5: A communication network. Nodes represent telephone numbers. Two nodes are connected if there has been a call between these two numbers. Some clusters of nodes are only visible in either the initial or the target drawing. Image 1 shows the initial drawing and image 4 shows the target drawing. Image 2 shows the nodes not visible in drawing 4 painted grey. Image 3 shows the nodes not visible in the initial drawing painted grey.

3.4.8 Scenario 6

Scenario 6 is taken from category B of the 2000 Graph Drawing competition. It displays a social network indicating how often members of a team in a company communicate with each other, with clients, and with domain experts. The graph is displayed in figure 3.13. Drawing 3.13.1 shows the graph with a circular layout. All nodes are placed on a circle around a virtual origin. Drawing 3.13.2 shows the same graph drawn using a force directed layout algorithm. The graph contains 100 nodes and 213 edges.

3.4.9 Scenario 7

Scenario 7 is a six bit shift-register graph and was featured in category B of the 1999 [14] graph drawing competition. The graph can also be seen as two intertwined trees. The graph contains 64 nodes and 126 edges. The two drawings of the graph in figure 3.14 show the result of applying traditional tree layout algorithms to the two trees contained in the graph. This scenario is especially challenging for methods based on regression analysis as many nodes are on the same line.

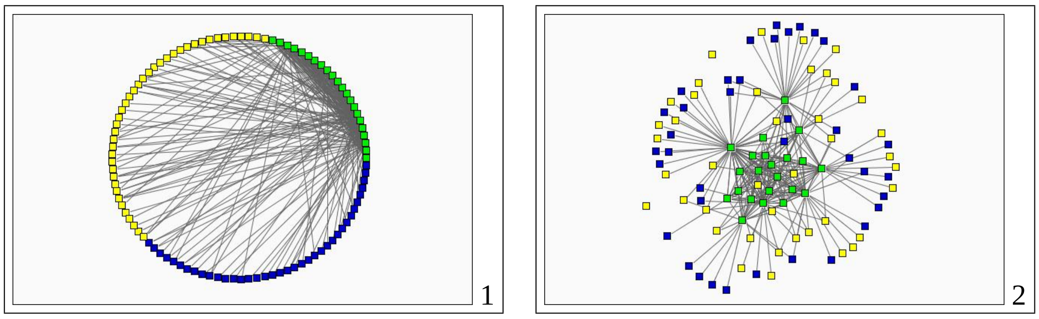


Figure 3.13: Scenario 6: Two different drawings of a graph which encodes a social network. Image 1 shows the graph drawn with a circular layout algorithm. Image 2 shows the same graph drawn by a force directed layout algorithm.

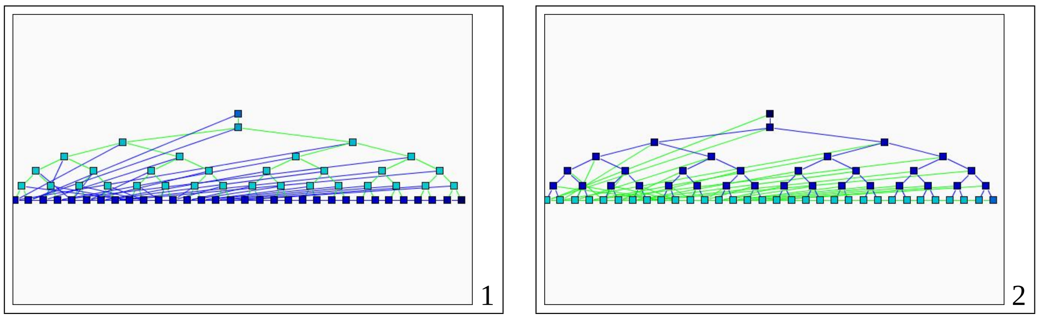


Figure 3.14: Scenario 7: The graph encodes a six bit shift-register. Both drawings were generated by a modified tree layout algorithm. Different nodes were used as roots of the “trees” in the two drawings.

3.4.10 Scenario 8

Scenario 8 features a fractal graph[2]. The drawing in figure 3.15.1 shows the graph drawn using a force directed layout algorithm. Figure 3.15.2 shows the graph drawn in a diagonal, fractal fashion. This scenario is challenging as edge lengths and relative node positions change significantly. Containing 2559 nodes and 4092 edges the graph is also comparatively large.

In the following chapters we use the introduced framework and the criteria discussed in section 3.2 to develop methods which compute animations using a variety of paradigms and foci. We evaluate these approaches analytically as well as measure their performance conducting the experiments outlined in section 3.4. We also compare the developed methods to existing approaches where possible.

[2]A fractal graph consists of a pattern that is recursively repeated over several levels

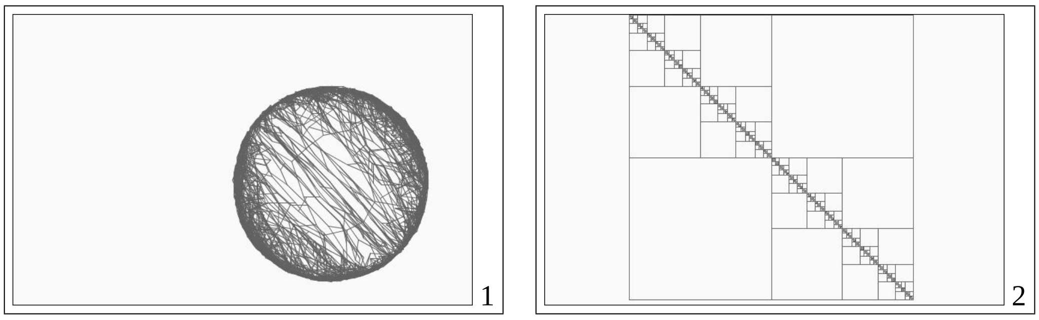


Figure 3.15: Scenario 8: The graph is artificial and does not encode any real world data. This scenario is especially challenging for animation algorithms as lengths and relative positions of nodes change significantly.

CHAPTER 4

Naive Methods for Graph Animations

Although graph drawing systems have been around for several decades, little serious research has been conducted in the area of general graph animations. Some naive or specialized techniques of animating changes in graph layouts however exist in several graph drawing tools, for example AGD [119] or D-ABDUCTOR [77]. In this chapter we introduce and investigate these approaches.

4.1 Direct Linear Interpolation

4.1.1 The Method

The most intuitive way to move an object from a point A to another point B is on the straight line connecting the two points. This approach has actually been implemented in some existing systems, for example AGD [119]. We can therefore formulate the following animation method between two key frames:

1. Remove nodes and edges which appear in the first frame but do not appear in the second frame.

   Several approaches are possible. Nodes could just disappear, slowly fade out, or move to some position outside the visible screen area. We do not discuss these strategies in detail and for our purposes assume that nodes and edges fade out as it appears to be a feasible and non-controversial method.

2. Move all nodes and edges on a straight line towards their target position

   Given a graph $G = (V, E)$ the position of each node $v \in V$ at time t during the animation is given as:

$$P_v(t) = (1-t) P_v(0) + tP_v(1)$$

The speed of the movement can be controlled by modifying t using a regulator function as described in section 3.3 on page 87.

3. Add nodes and edges which did not exist in the first frame, but do exist in the second frame.

   This step is equivalent to step 1 and similar approaches are possible. Nodes could just appear, slowly fade in, or move to to their positions from somewhere outside the visible screen area. For our purposes we assume that nodes and edges fade in as it appears to be a feasible and non-controversial method.

This procedure can be repeated between each pair of key frames in a multi-key-frame scenario. As a variation, the node positions can be interpolated using spline curves if more than two key frames are specified.

4.1.2 Analysis

1. Time to initialize the animation (Init)

   The *Direct Linear Interpolation* method is very static and no pre-processing is necessary in a two key-frame scenario. All necessary computations can be made during the animation. In multi-key-frame scenarios where is spline interpolation is used to compute the node paths, it is sensible to compute the spline curves only once in the beginning. The computational cost in such cases is linear in the number of key frames and the number of nodes. Assuming a constant number of key frames the *Direct Linear Interpolation* method is sufficiently fast, as defined in section 3.2 from page 74.

2. Time to compute each frame (T/F)

   The computational cost for each frame is insignificant. The position of each node in a frame can be computed with a small constant number of computations.

3. Static edge crossings (SEC), constant edge length (CEL), temporary edge crossings (TEC), unnecessary node intersections (UNI)

   The *Direct Linear Interpolation* method does not explicitly consider these criteria. It is not difficult to construct scenarios which exhibit arbitrarily bad behavior for these measures. For example, in the case of a uniform 180^o rotation of a graph $G = (V, E)$ around its center, all nodes intersect at some point during the animation. The resulting value for *unnecessary node intersections* (UNI) is $\frac{|V|(|V|-1)}{2}$. Similar examples for the other measures exist.

4. Smoothness

   The movement is smooth[1] between two key frames. Using linear interpolation the movement is generally not smooth over multiple key frames, with an abrupt change of direction expected at each key frame. Using spline interpolation a smooth motion over all key frames can be guaranteed.

5. Uniform node movement (UNM)

   If the underlying transformation of the graph is very simple, for example a uniform translation of the whole graph, the *Direct Linear Interpolation* method achieves the optimal value 0 for *uniform node movement* (UNM). However, for more complex motions, such as rotations, this is no longer true. Scenario 2 on page 89 illustrates this. In scenario 2, all nodes could theoretically move uniformly to their target positions by performing a rotation around the center of the graph. The *Direct Linear Interpolation* method, however, is not able to find or describe this particular kind of motion and achieves bad results for *uniform node movement*. More generally, let us consider the general case of a 180^o rotation of a graph $G = (V, E)$ around its center. To simplify calculations, let us further assume without loss of generality that the center of the graph is located at the origin. Performing a pure 180^o rotation of the whole graph around its center would result in the optimal value zero for *uniform node movement*. We now investigate the value for *uniform node movement* that we get if we apply the *Direct Linear Interpolation* method to this scenario. The *uniform node movement* is defined as:

$$unm = \sum_{u,v \in V} \int_0^1 \left(d_{(u,v)}(t) - \left((1-t)\, d_{(u,v)}(0) + t d_{(u,v)}(1)\right)\right)^2 dt.$$

   As the distance between nodes in a pure rotation is constant we can simplify this expression to:

[1]For the definition of *smooth* as it is given in section 2.3.5 on page 50.

$$unm = \sum_{u,v\in V} \int_0^1 \left(d_{(u,v)}(t) - d_{(u,v)}(0)\right)^2 dt.$$

The underlying transformation is a 180^o rotation of the whole graph. The x and y coordinates of each node $v \in V$ during the animation are thus given by:

$$\begin{aligned} X_v(t) &= (1-t)\,X_v(0) + t\,(-X_v(0)) \\ Y_v(t) &= (1-t)\,Y_v(0) + t\,(-Y_v(0))\,. \end{aligned}$$

Substituting X_v and Y_v in unm, we get:

$$unm = \sum_{u,v\in V} \int_0^1 \left(\sqrt{(X_v(t) - X_u(t))^2 + (Y_v(t) - Y_u(t))^2} - d_{(u,v)}(0)\right)^2 dt$$

this yields:

$$\begin{aligned} unm &= \sum_{u,v\in V} \frac{4}{3}\Big(-2X_u(0)\,X_v(0) - 2Y_u(0)\,Y_v(0) + X_u(0)^2 + Y_u(0)^2 + \cdots \\ &\quad \cdots X_v(0)^2 + Y_v(0)^2\Big) \\ &= \frac{4}{3} \sum_{u,v\in V} d_{(u,v)}(0)^2 \end{aligned}$$

This means that we can provoke arbitrarily bad behavior by choosing initial node positions which are far apart from each other. Furthermore, as the distances between all pairs of nodes are considered, *uniform node movement* grows quadratically in respect to the number of nodes in the graph.

6. Node path length (NPL)

When using the *Direct Linear Interpolation* method, *node path length* is always minimal, as all nodes move on the shortest possible path. When using spline interpolation, we can expect an essential tradeoff between *node path length* and the degree of smoothness. The higher the derivability of a function, the higher the likelihood that it contains unwanted "spikes"[2] which, unnecessarily, increase the node path length.

4.1.3 Empirical Evaluation

In this section we describe the performance of the *Direct Linear Interpolation* animation technique on our benchmark scenarios. For each scenario we give nine snapshots from the actual animation. Further we give some informal remarks about the performance of the method where appropriate.

We also give a table and a bar chart graphic which display the recorded measures for the currently discussed method, as well as the median and the best result for each measure over all methods. The values in the bar chart are normalized to the range zero to one with the worst result for each measure over all methods being scaled to one. This allows a rough classification of the method in respect to the other methods. A detailed comparison of the methods can be found in chapter 7. It should be noted that results in the row *best* are the independent best results for each measure and not necessarily the results of the overall best method. In particular, this also means that there might not even exist a method which can achieve all best results.

The electronic version of this thesis has the full animations embedded. Using an appropriate display program, for example Adobe Acrobat Reader(TM), the animations can be started by clicking on any of the nine snapshots.

[2]For a discussion of spikes in point interpolations see section 2.3.5 on page 50

Scenario 1

Figure 4.1 shows the resulting animation when applying the *Direct Linear Interpolation* method to scenario 1. The numerical results of the animation in regard to our measures are summarized in the

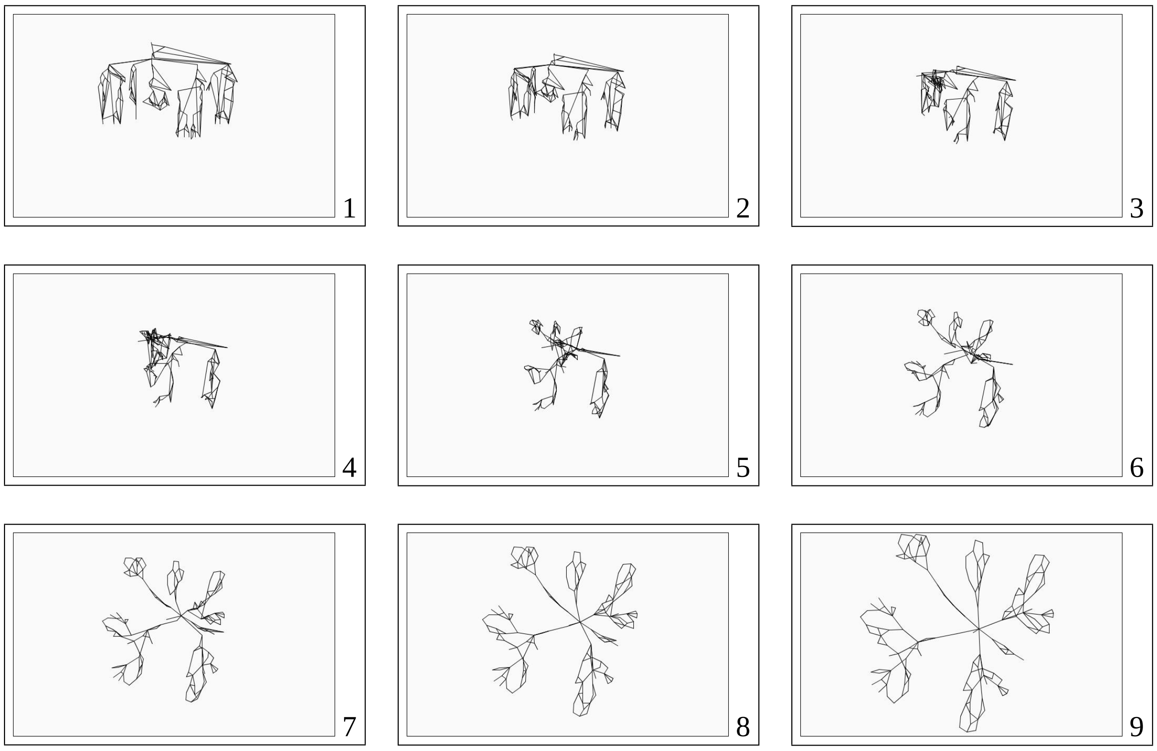
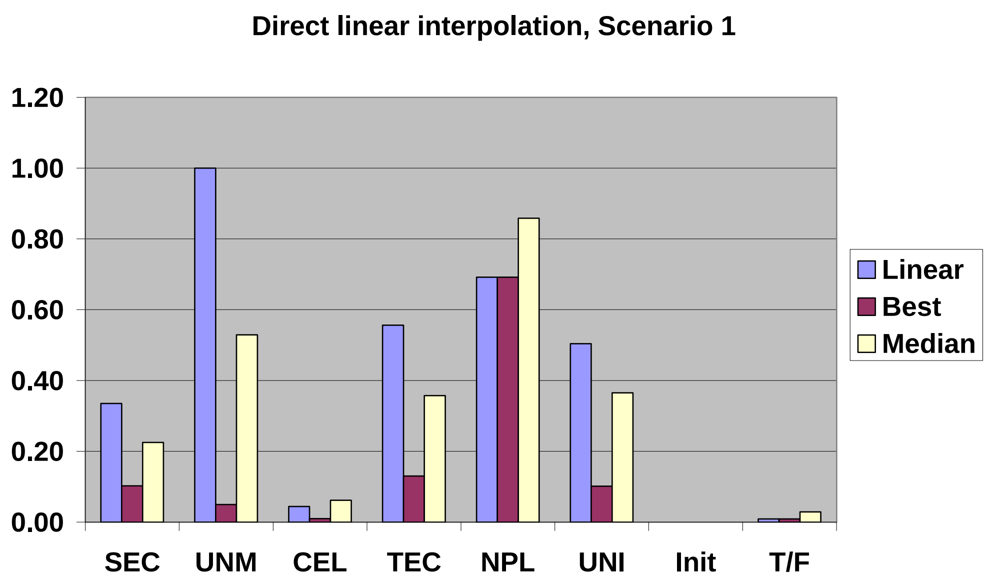


Figure 4.1: This figure shows 9 snapshots of a 101 frame animation. The animation is the result of applying the *Direct Linear Interpolation* method to scenario 1.

following table:

Method	SEC	UNM	CEL	TEC	NPL	UNI	Init	T/F
Linear	130	5.020E+09	1,843,276	2,095	347,983	129	0	0.5
Best	40	2.484E+08	402,170	490	347,983	26	0	0
Median	87	2.656E+09	2,577,963	1,346	431,802	94	0	1.6



As expected, the *Direct Linear Interpolation* method achieves optimal values regarding computation time (Init, T/F) and *node path length* (NPL). For all other measures it is significantly worse than the best method. For all measures but *constant edge length* (CEL) it behaves, in some cases significantly, worse than the median. It seems surprising that we see a better than median behavior regarding *constant edge length* but not regarding *uniform node movement* (UNM). This behavior can be explained by the fact that some methods try to optimize *uniform node movement* without specifically considering edges. During animations where nodes which are connected by an edge are assigned to different groups of uniformly moving nodes, these methods can still achieve good values for *uniform node movement* but achieve bad values for *constant edge length*.

Scenario 1b

Figure 4.2 shows the result of applying the *Direct Linear Interpolation* method to scenario 1b. The

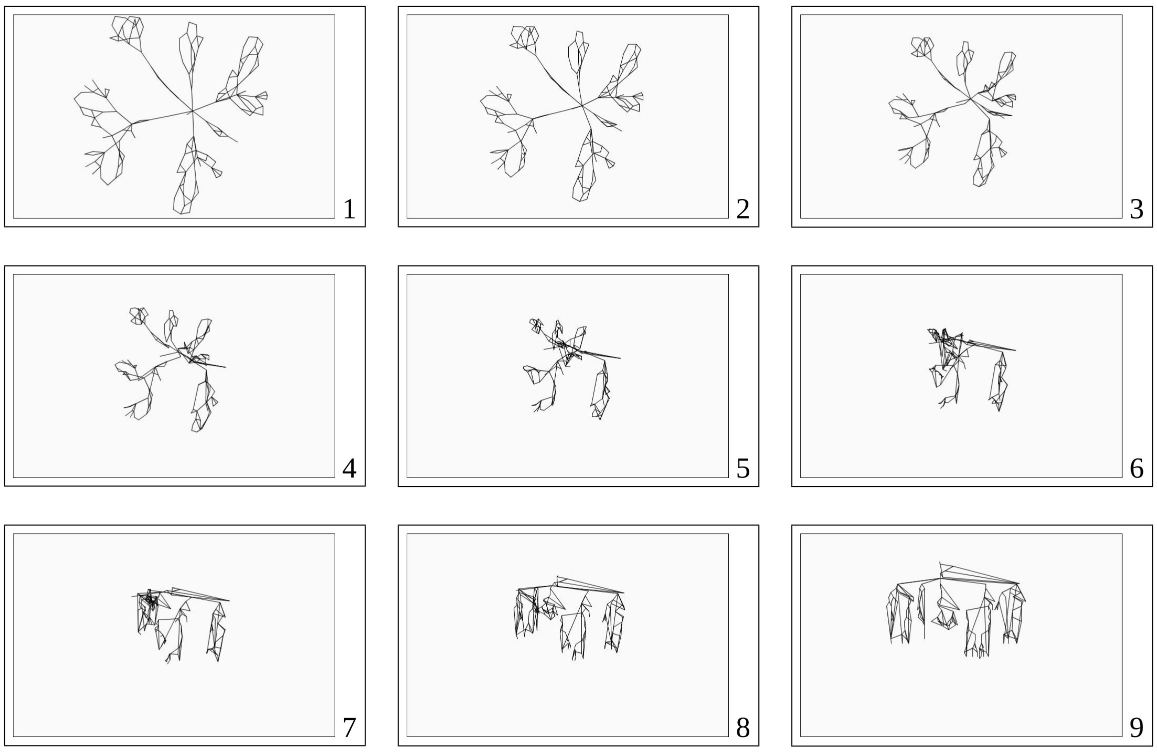
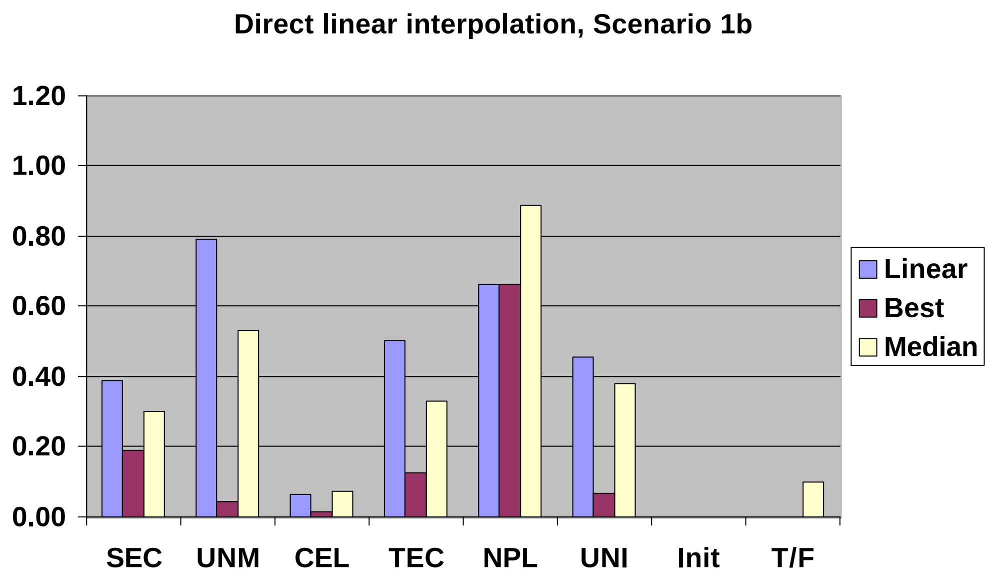


Figure 4.2: This figure shows 9 snapshots of a 101 frame animation. The animation is the result of applying the *Direct Linear Interpolation* method to scenario 1b.

numerical results of the animation in regard to our measures are summarized in the following table:

Method	SEC	UNM	CEL	TEC	NPL	UNI	Init	T/F
Linear	130	5.020E+09	1,843,276	2,138	347,983	129	0	0.0
Best	64	2.697E+08	395,880	533	347,983	19	0	0
Median	100	3.371E+09	2,146,880	1,409	465,534	108	0	5.1



Scenario 1b is identical to scenario 1 with exchanged initial and target drawing. As the *Direct Linear Interpolation* method is symmetrical, we see the same results as before. Small differences, for example for *temporary edge crossings* (TEC), are due to the discrete approximation of some measures.

Scenario 2

Figure 4.3 shows the result of applying the *Direct Linear Interpolation* method to scenario 2. The

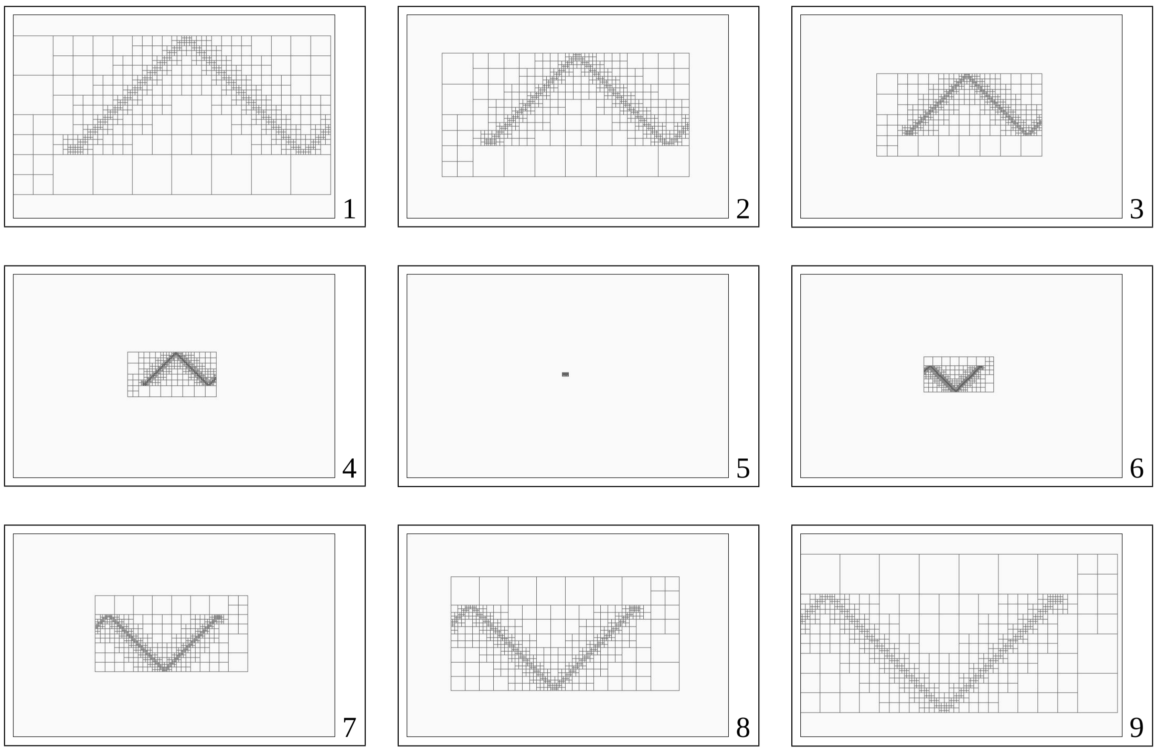
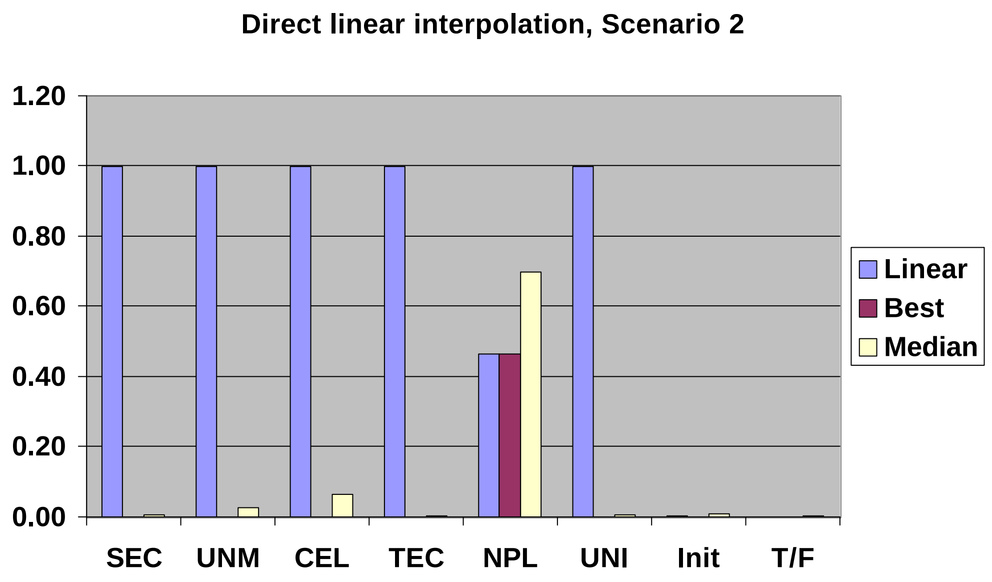


Figure 4.3: This figure shows 9 snapshots of a 101 frame animation. The animation is the result of applying the *Direct Linear Interpolation* method to scenario 2.

numerical results of the animation in regard to our measures are summarized in the following table:

Method	SEC	UNM	CEL	TEC	NPL	UNI	Init	T/F
Linear	12,891	1.675E+10	212,992	1,301,941	440,110	3,813,032	50	1.6
Best	4	1.066E-20	0	374	440,110	0	0	1
Median	92	4.259E+08	13,608	3,468	662,794	24,579	270	4.7



The *Direct Linear Interpolation* method show exceptionally bad results for scenario 2 regarding most measures. Theoretically, scenario 2 can be animated with the optimal value zero for all measures but *node path length* (NPL) and time (Init,T/F) by rotating the whole graph by 180^o around its center. Rotations by large angles are a general problem for the *Direct Linear Interpolation* method. The intermediate drawings are always distorted to a degree which is relative to the angle of the rotation. In the case of a 180^o rotation, the graph is even collapsed to a single point at one stage. This can be seen in figure 4.3.5.

Scenario 2b

Figure 4.4 shows the result of applying the *Direct Linear Interpolation* method to scenario 2b. The

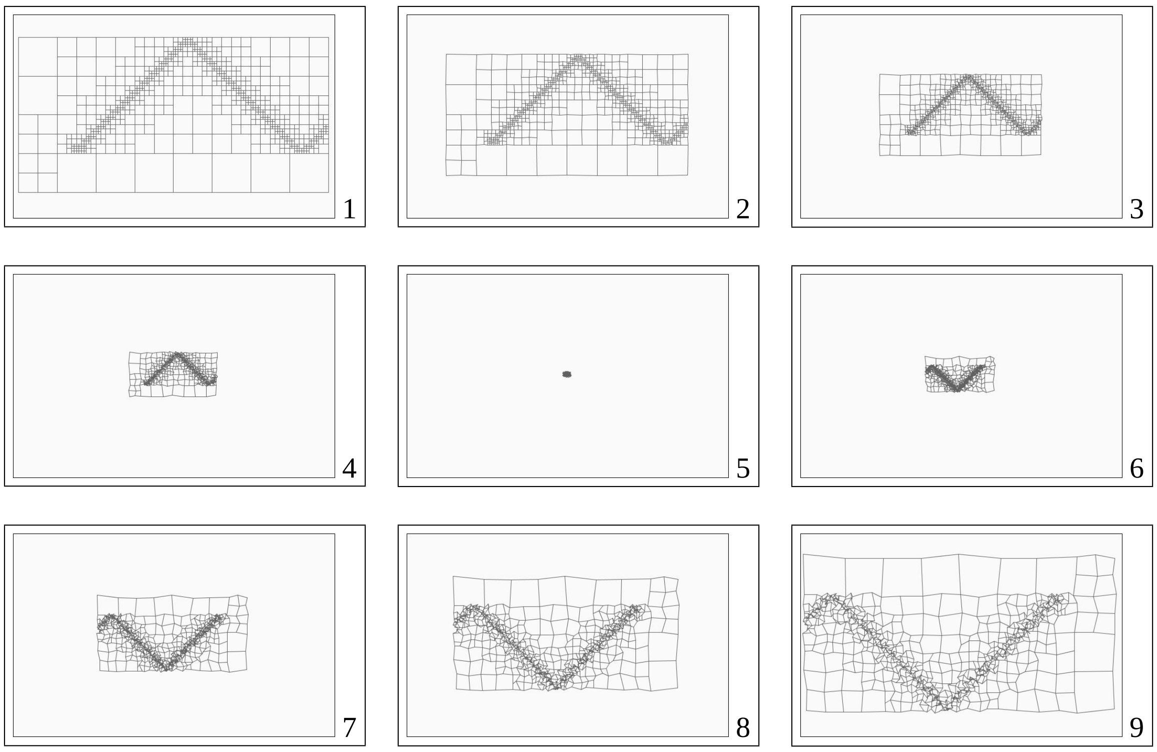
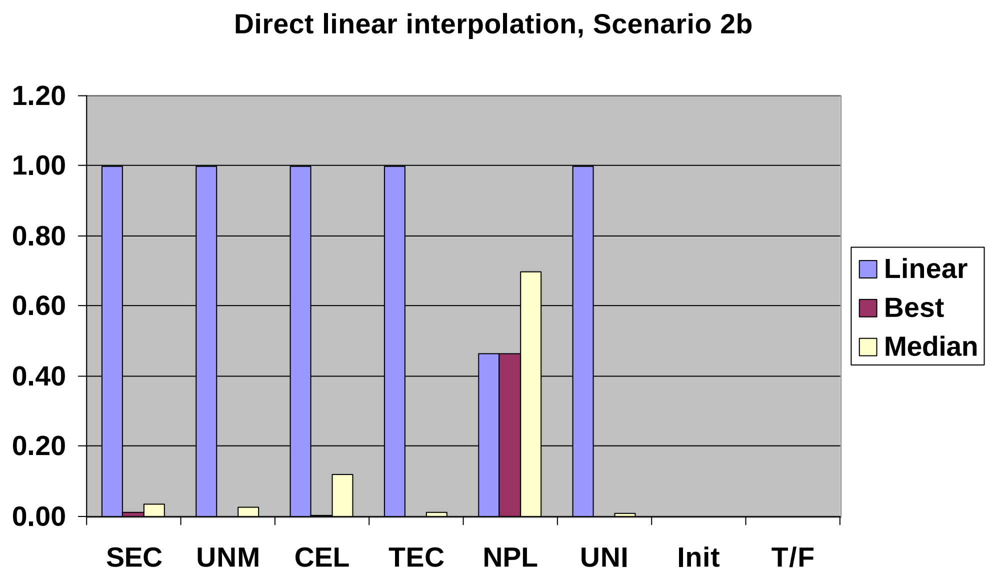


Figure 4.4: This figure shows 9 snapshots of a 101 frame animation. The animation is the result of applying the *Direct Linear Interpolation* method to scenario 2b.

numerical results of the animation in regard to our measures are summarized in the following table:

Method	SEC	UNM	CEL	TEC	NPL	UNI	Init	T/F
Linear	9,681	1.670E+10	202,046	582,018	439,723	3,577,834	0	1.0
Best	105	6.805E+03	750	612	439,723	749	0	1
Median	347	4.339E+08	24,396	6,301	662,911	33,607	30	4.3



Scenario 2b is similar to scenario 2. Again the transformation of the graph is close to a 180^o rotation and the *Direct Linear Interpolation* method shows the same typical graph distortions.

Scenario 3

Figure 4.5 shows the result of applying the *Direct Linear Interpolation* method to scenario 3. The

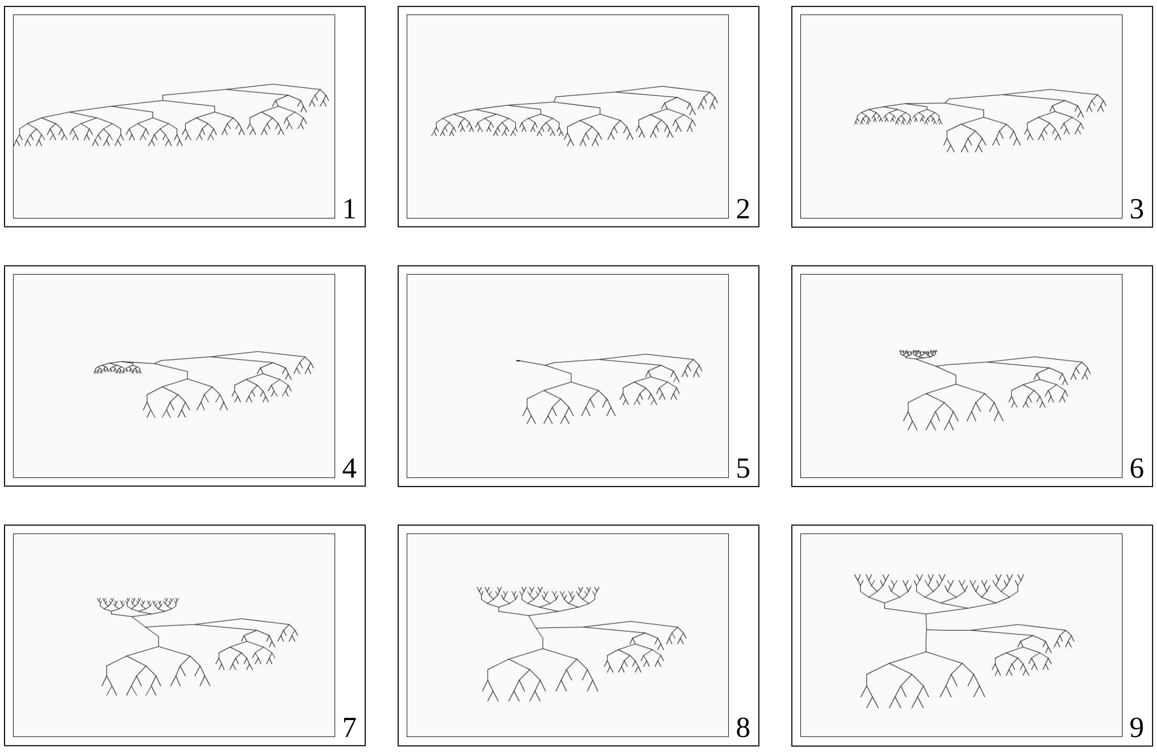
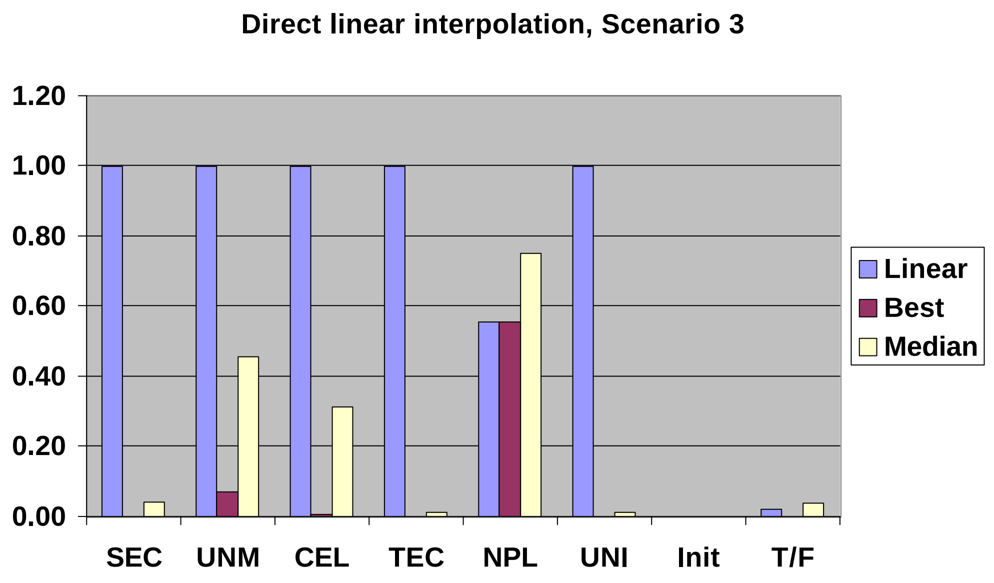


Figure 4.5: This figure shows 9 snapshots of a 101 frame animation. The animation is the result of applying the *Direct Linear Interpolation* method to scenario 3.

numerical results of the animation in regard to our measures are summarized in the following table:

Method	SEC	UNM	CEL	TEC	NPL	UNI	Init	T/F
Linear	40	3.962E+08	114,892	4,055	94,772	23,519	0	0.6
Best	0	2.753E+07	720	0	94,772	0	0	0
Median	2	1.799E+08	35,772	51	128,102	245	0	1.1



The transformation described in scenario 3 can theoretically be animated without a single edge crossing or node intersection. Again, the *Direct Linear Interpolation* method achieves rather poor results due to a large rotational component in the transformation of one sub graph.

Scenario 4

Figure 4.6 shows the result of applying the *Direct Linear Interpolation* method to scenario 4. The

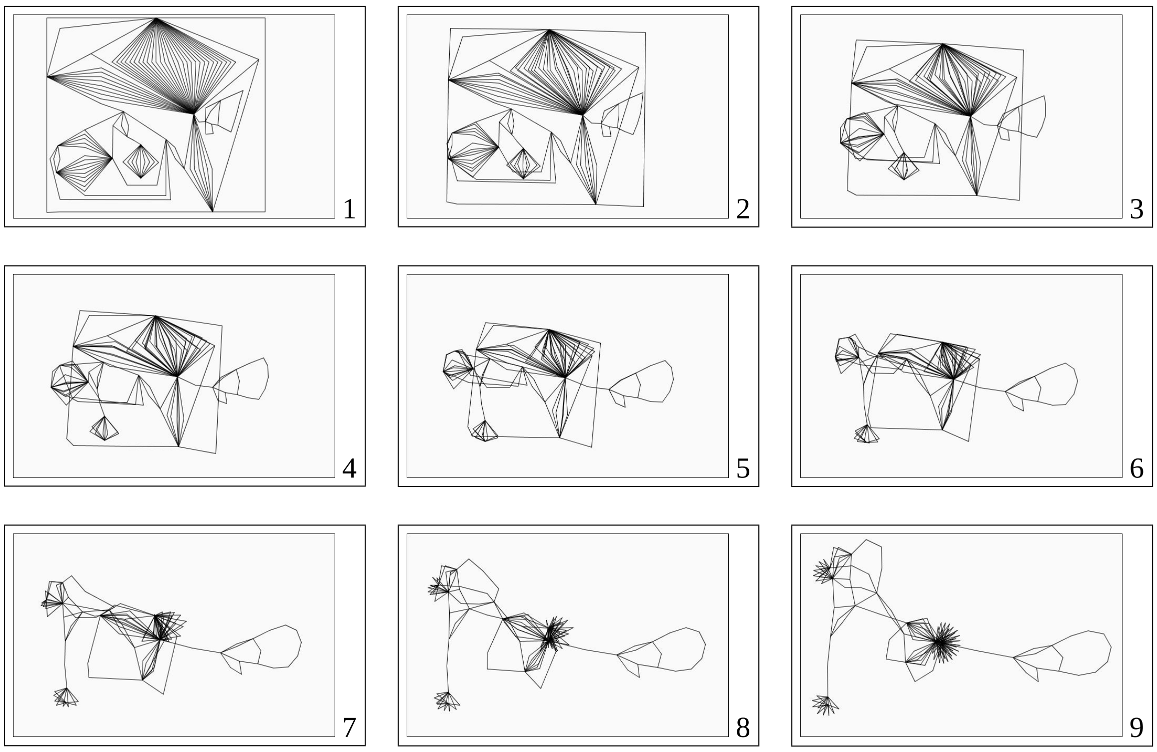
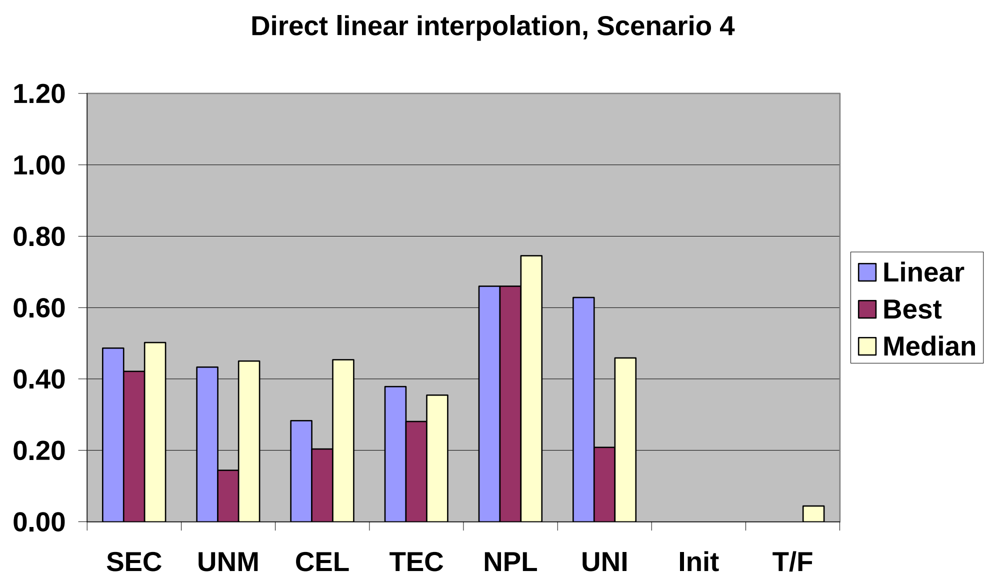


Figure 4.6: This figure shows 9 snapshots of a 101 frame animation. The animation is the result of applying the *Direct Linear Interpolation* method to scenario 4.

numerical results of the animation in regard to our measures are summarized in the following table:

Method	SEC	UNM	CEL	TEC	NPL	UNI	Init	T/F
Linear	190	2.877E+08	1,326,461	1,328	120,561	169	0	0.0
Best	164	9.562E+07	954,237	985	120,561	56	0	0
Median	196	2.991E+08	2,125,756	1,244	136,133	124	0	0.5



The *Direct Linear Interpolation* method achieves comparatively good results in this scenario. In all categories it is better or almost as good as the median. It is significantly worse than the best algorithm in regard to *constant edge length* (CEL), however.

Scenario 5

Figure 4.7 shows the result of applying the *Direct Linear Interpolation* method to scenario 5. The

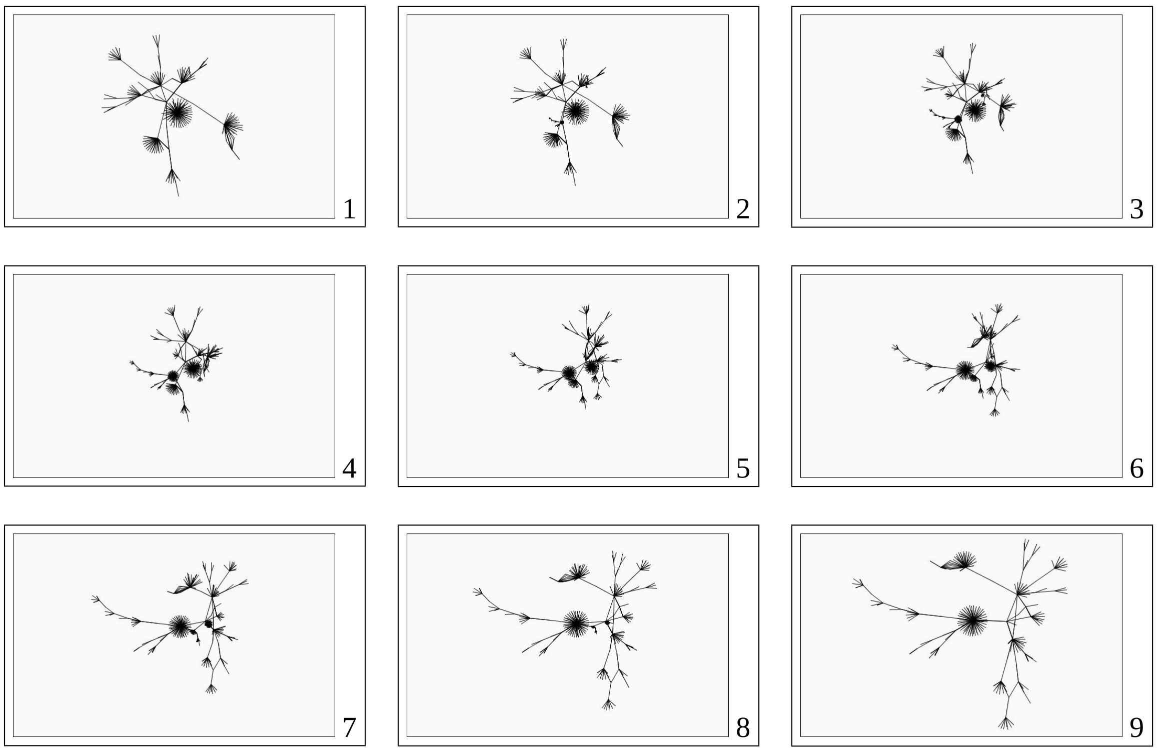
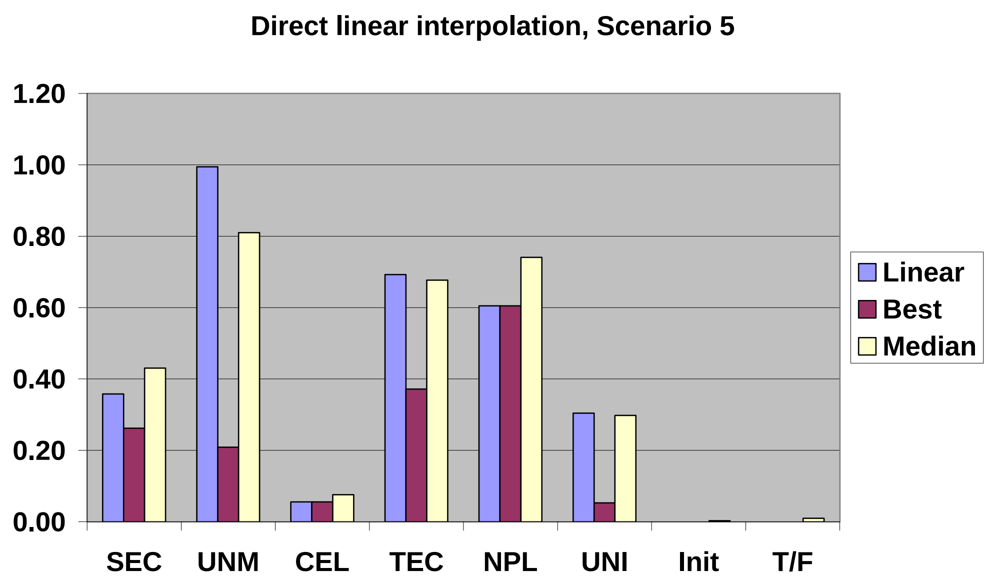


Figure 4.7: This figure shows 9 snapshots of a 101 frame animation. The animation is the result of applying the *Direct Linear Interpolation* method to scenario 5.

numerical results of the animation in regard to our measures are summarized in the following table:

Method	SEC	UNM	CEL	TEC	NPL	UNI	Init	T/F
Linear	420	1.470E+10	2,399,401	15,725	597,271	277	0	0.0
Best	308	3.088E+09	2,399,401	8,438	597,271	48	0	0
Median	505	1.198E+10	3,262,048	15,373	731,150	272	55	1.1



Although the *Direct Linear Interpolation* method achieves results which are slightly better or at least not significantly worse than the median for all measures, the *Direct Linear Interpolation* method nevertheless fails to score close to best results in any category but the obligatory *node path length* (NPL) and time (Init, T/F).

Scenario 6

Figure 4.8 shows the result of applying the *Direct Linear Interpolation* method to scenario 6. The

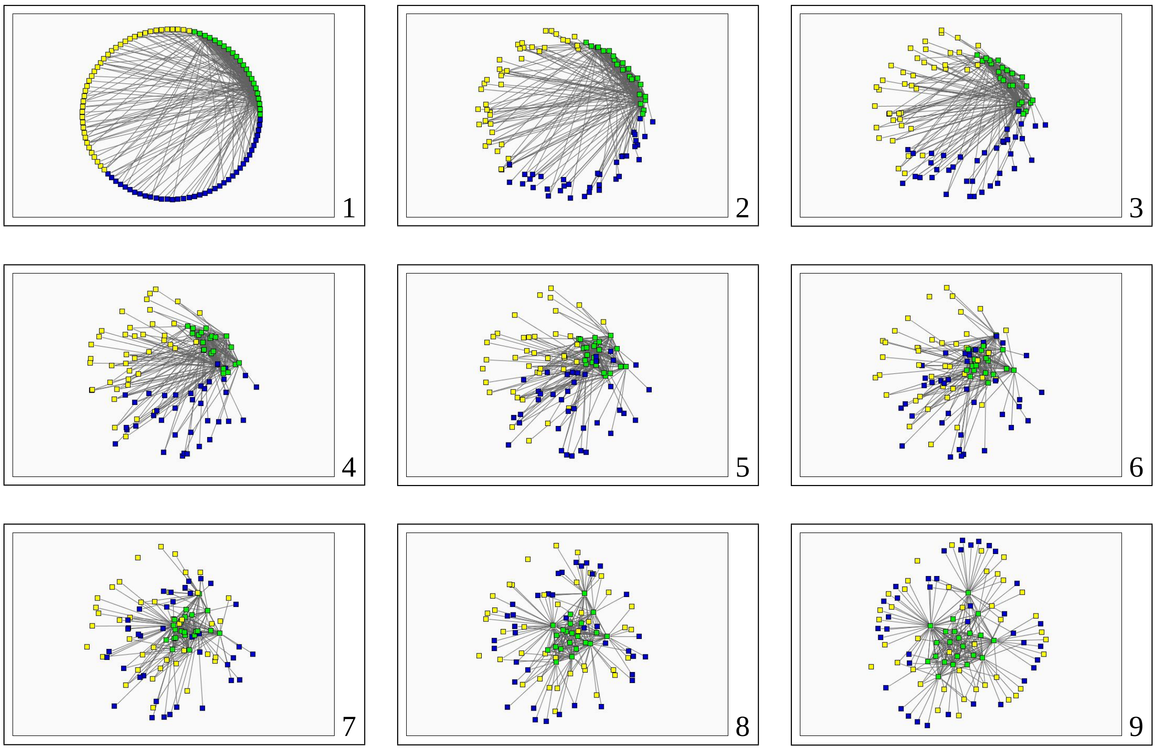
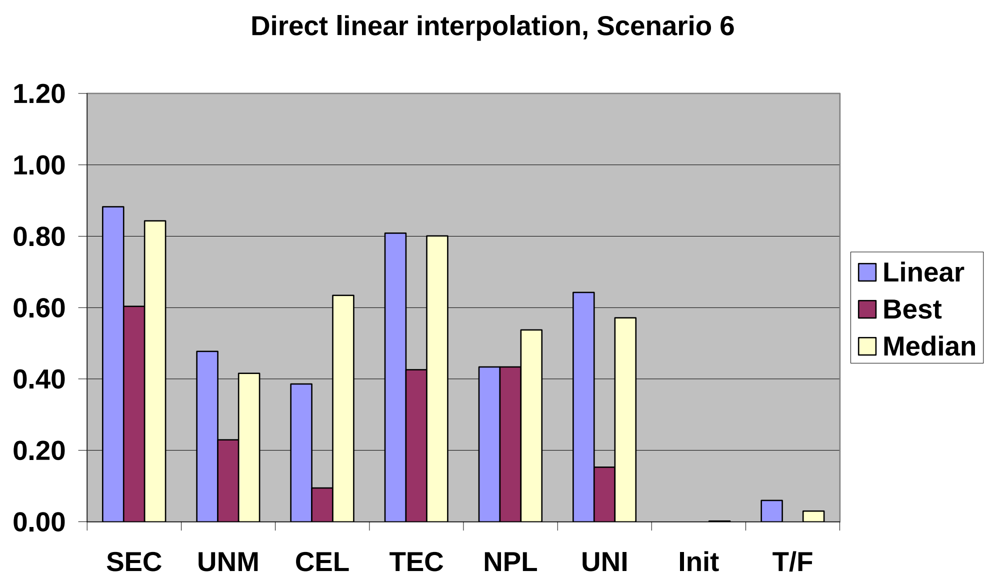


Figure 4.8: This figure shows 9 snapshots of a 101 frame animation. The animation is the result of applying the *Direct Linear Interpolation* method to scenario 6.

numerical results of the animation in regard to our measures are summarized in the following table:

Method	SEC	UNM	CEL	TEC	NPL	UNI	Init	T/F
Linear	7,095	2.385E+08	4,174,533	15,762	73,957	185	0	0.5
Best	4,853	1.147E+08	1,020,039	8,301	73,957	44	0	0
Median	6,777	2.077E+08	6,856,478	15,610	91,672	165	25	0.2



The *Direct Linear Interpolation* method achieves close to median but far from optimal results for all measures but *node path length* and time.

Scenario 7

Figure 4.9 shows the result of applying the *Direct Linear Interpolation* method to scenario 7. The

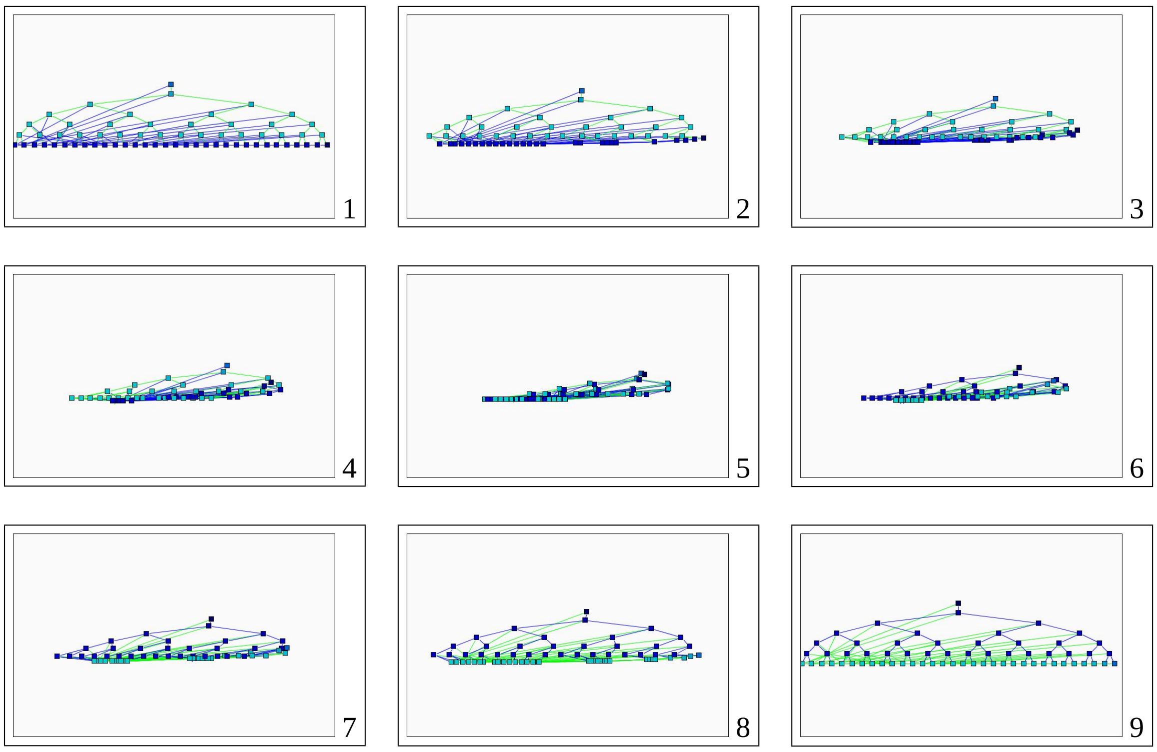
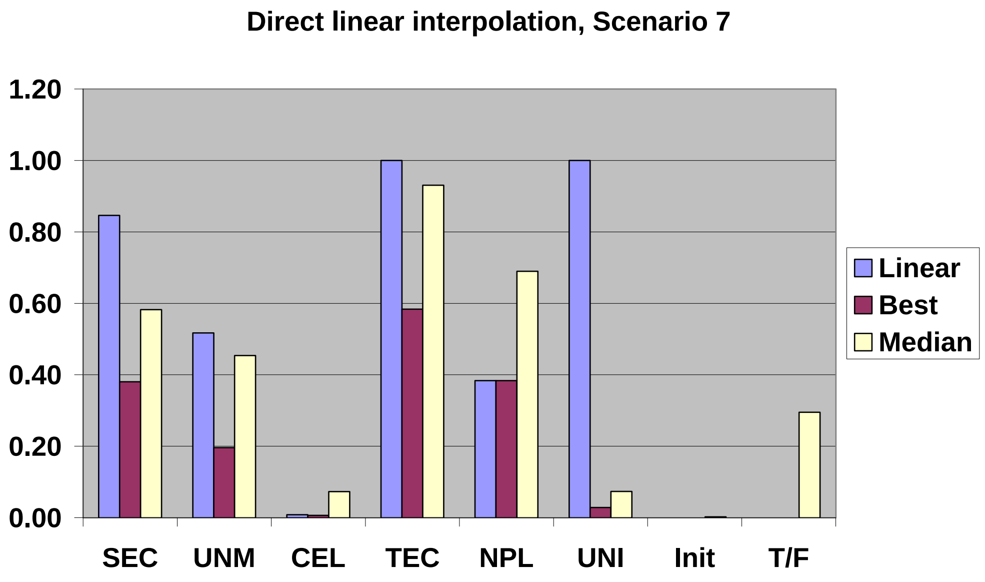


Figure 4.9: This figure shows 9 snapshots of a 101 frame animation. The animation is the result of applying the *Direct Linear Interpolation* method to scenario 7.

numerical results of the animation in regard to our measures are summarized in the following table:

Method	SEC	UNM	CEL	TEC	NPL	UNI	Init	T/F
Linear	1,011	4.858E+07	90,948	4,975	25,170	2,692	0	0.0
Best	454	1.841E+07	71,857	2,905	25,170	77	0	0
Median	696	4.262E+07	816,052	4,631	45,283	198	25	1.1



The *Direct Linear Interpolation* method is significantly better than the median regarding *constant edge length* (CEL) in scenario 7, but also significantly worse than the median regarding *unnecessary node intersections* (UNI). This indicates an essential tradeoff between these to measures in this scenario. This hypothesis is further supported by the results for the other methods. It is not obvious by watching the animations if *constant edge length* or *unnecessary node intersections* is the perceptually more important measure in this case. The author tends to favor the animation with good values for *unnecessary node intersections* over the solution with good values for *constant edge length*, but this is only a subjective opinion and not based on scientific results. It would be necessary to conduct extensive HCI studies to prove that this bias is actually a common tendency.

Scenario 8

Figure 4.10 shows the result of applying the *Direct Linear Interpolation* method to scenario 8. The

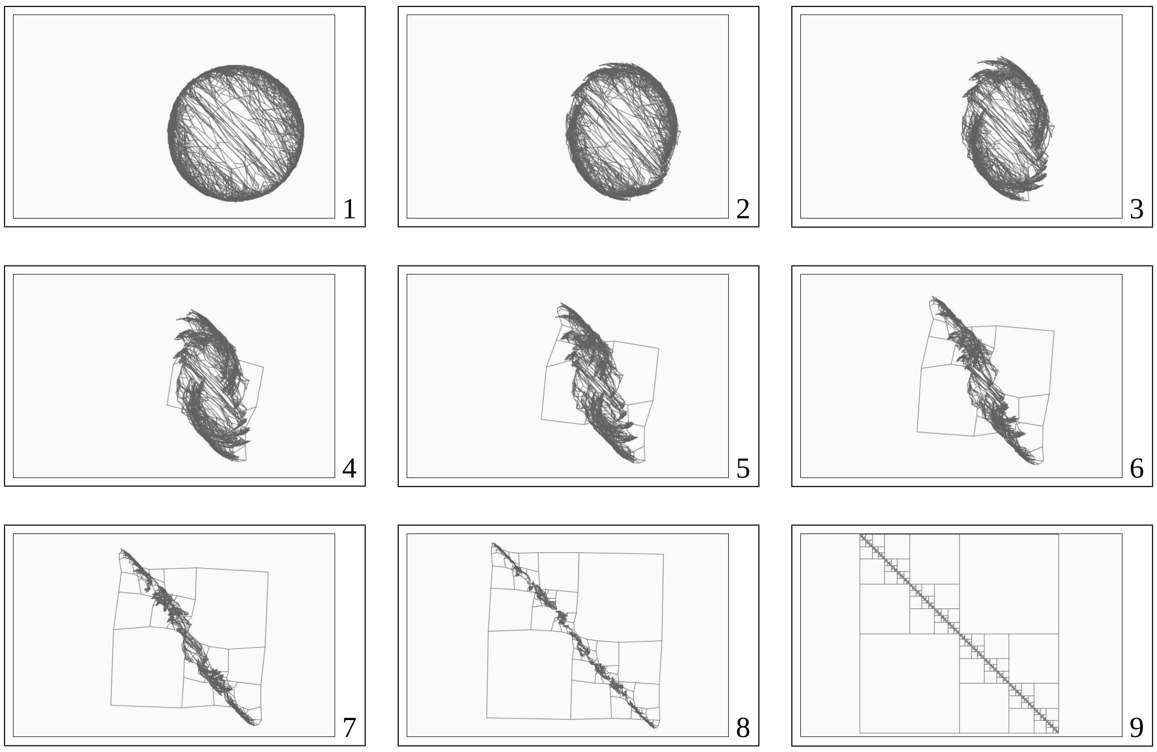
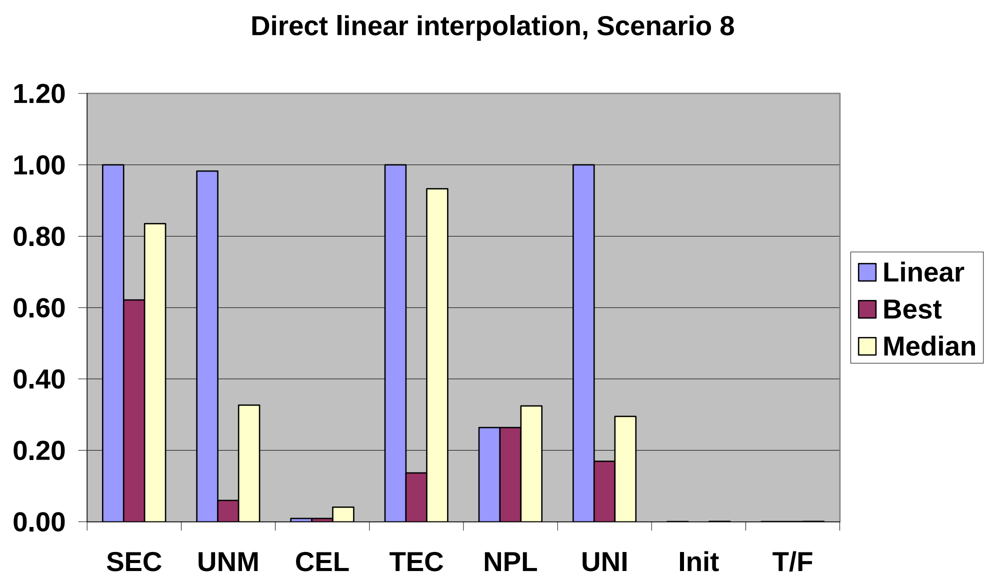


Figure 4.10: This figure shows 9 snapshots of a 101 frame animation. The animation is the result of applying the *Direct Linear Interpolation* method to scenario 8.

numerical results of the animation in regard to our measures are summarized in the following table:

Method	SEC	UNM	CEL	TEC	NPL	UNI	Init	T/F
Linear	46,185	2.829E+09	4,138	2,112,739	290,510	7,427,591	50	7.2
Best	28,706	1.718E+08	4,138	289,169	290,510	1,256,611	0	4
Median	38,574	9.407E+08	18,486	1,971,003	357,359	2,192,830	80	13.7



We note significantly bad values for *unnecessary node intersections* (UNI) and *uniform node movement* (UNM). The latter is in notable contrast to the good values for *constant edge length* (CEL), where direct linear interpolation even achieves the best results of all methods. Again we can deduce an essential tradeoff, especially as a comparison with other methods shows that they also have either good values for *uniform node movement* or *or constant edge length*, but not for both.

4.1.4 Summary

The *Direct Linear Interpolation* method produces results for many scenarios which are close to the median. By definition, it always achieves optimal values for *node path length* (NPL) and thus guarantees a short animation. The method requires an insignificant amount of time for preprocessing and for computing each frame.

In cases which contain structured transformations, especially transformation with large rotational components, the *Direct Linear Interpolation* method generally achieves very poor results for *static edge crossings* (SEC), *temporary edge crossings* (TEC), *uniform node movement* (UNM), *constant edge length* (CEL), and *unnecessary node intersections* (UNI).

The *Direct Linear Interpolation* method seems to be an appropriate animation method when little structure in the underlying transformations is expected. However, if such structure exists, then other methods are generally preferable.

4.2 Orthogonal Interpolation

4.2.1 The Method

The *Orthogonal Interpolation* method is similar to the *Direct Linear Interpolation* method. The frames of the animation are computed by directly interpolating the differences between the initial positions and the target positions of the nodes of the graph. In contrast to the *Direct Linear Interpolation* method, however, the x coordinates and the y coordinates are interpolated independently. That is, first the x coordinates are interpolated from the initial positions to the target positions and subsequently the y coordinates are interpolated. Although this approach seems to be quite similar to the direct interpolation approach, we found that the visual impression can be significantly different. The human brain tends to interpret parallel movements of all parts of an object as a three-dimensional rotation of the object in space. This has the effect that animations which are computed by using the *Orthogonal Interpolation* method often create an optical illusion. In this illusion the transformation of the graph is perceived as the rotation of a three dimensional graph instead of a changing two dimensional graph. Although it is unclear if this actually helps in understanding the changes, the resulting animation in most cases appears “aesthetically pleasing”. The purpose of this section is to formally introduce this method and evaluate its properties against our measures.

Given a graph $G = (V, E)$ the position of a node $v \in V$ at time t is given by:

$$P_v(t) = \begin{cases} \left(\pi_1\left(P_v(0)\right), \pi_2\left(2tP_v(0) + P_v(1)(1-2t)\right)\right) & \text{for } 0 \le t \le \frac{1}{2}, \\ \left(\pi_1\left(2\left(t-\frac{1}{2}\right)P_v(0) + P_v(1)\left(1-2\left(t-\frac{1}{2}\right)\right)\right), \pi_2\left(P_v(1)\right)\right) & \text{otherwise.} \end{cases}$$

This procedure can be repeated between each pair of key frames in a multi-key-frame scenario.

4.2.2 Analysis

The analysis of most measures in regard to the *Orthogonal Interpolation* method gives the same results as for the *Direct Linear Interpolation* method (Section 4.1.2 on page 97). We therefore only mention those measures here for which the *Orthogonal Interpolation* method behaves differently.

1. Smoothness

   The movement is piecewise smooth. During the interpolation along each axis the motion is smooth. At the point where the interpolated axis is changed, however, the direction of motion turns abruptly.

2. UNM

   It is almost impossible for the *Orthogonal Interpolation* method to achieve the optimal value zero for *uniform node movement* (UNM):

   $$unm = \sum_{u,v \in V} \int_0^1 \left(d_{(u,v)}\left(t\right) - \left(\left(1-t\right)d_{(u,v)}\left(0\right) + td_{(u,v)}\left(1\right)\right)\right)^2 dt.$$

   The function

   $$f\left(t\right) = \left(d_{(u,v)}\left(t\right) - \left(\left(1-t\right)d_{(u,v)}\left(0\right) + td_{(u,v)}\left(1\right)\right)\right)^2$$

   is non-negative and continuous. Thus, for the sum of the integrals to be 0, $f\left(t\right)$ must be 0 for all $u, v \in V$ and $t \in [0 \ldots 1]$. Let us consider the case $t = 0.5$ for fixed u and v and $d\left(t\right) = d_{(u,v)}\left(t\right)$:

   $$\begin{aligned}
   0 &= \left(d\left(t\right) - \left(\left(1-t\right)d\left(0\right) + td\left(1\right)\right)\right)^2 \\
   0 &= \left(d\left(t\right) - \left(\left(1-t\right)d\left(0\right) + td\left(1\right)\right)\right)^2 \\
   0 &= d\left(t\right) - \left(\left(1-t\right)d\left(0\right) + td\left(1\right)\right) \\
   td\left(0\right) + td\left(1\right) &= d\left(t\right) \\
   0.5d\left(0\right) + 0.5d\left(1\right) &= d\left(0.5\right) \\
   0.5d\left(0\right) + 0.5d\left(1\right) &= \sqrt{(X_v(0) - X_u(0))^2 \cdots} \\
   & \overline{\cdots + (2 * 0.5Y_u(0) + (1 - 2 * 0.5)Y_u(1)) \cdots} \\
   & \overline{\cdots - (2 * 0.5Y_v(0) + (1 - 2 * 0.5)Y_v(1)))^2} \\
   0.5d\left(0\right) + 0.5d\left(1\right) &= \sqrt{\left(X_v\left(0\right) - X_u\left(o\right)\right)^2 + \left(Y_u\left(0\right) - Y_v\left(0\right)\right)^2} \\
   0.5d\left(0\right) + 0.5d\left(1\right) &= d\left(0\right) \\
   d\left(1\right) &= d\left(0\right)
   \end{aligned}$$

   We have shown that $d_{(u,v)}\left(1\right) = d_{(u,v)}\left(0\right)$ is a necessary precondition for the *Orthogonal Interpolation* method to be able to achieve zero for *uniform node movement*. Only two continuous, isometric transformations exist: rotation and translation of the whole graph. It is clear that a rotation cannot be described by the *Orthogonal Interpolation* method. The *Or-*

thogonal Interpolation method can therefore only achieve optimal values for *uniform node movement* in the special case of a uniform translation of the whole graph.

On the other hand, it is not difficult to provoke arbitrarily bad behavior. We consider the following simple example. The graph consists of two nodes v_1 and v_2. Node v_1 is positioned at the same coordinates in the initial and the target drawing and therefore does not move during the animation. Node v_2 only changes its x coordinate between the initial and target drawing and therefore does not move during the first half of the animation. For the first half of the animation the distance between the two nodes thus stays constant although it should change according to the measure. By choosing appropriate initial and target positions arbitrarily bad values for *uniform node movement* can be provoked.

3. NPL

   When using the *Orthogonal Interpolation* method, *node path length* is at most $\sqrt{2}$ times the minimal node path legnth for each node.

4.2.3 Empirical Evaluation

In this section we describe the performance of the *Orthogonal Interpolation* animation technique on our benchmark suit. For each scenario we give nine snapshots from the actual animation, as well as a table and a bar chart graphic displaying the results for the eight measures described in section 3.2 from page 74. The table and the graphic display the results for the currently discussed method, as well as the median results and the best results. We further include the results of the *Direct Linear Interpolation* method from section 4.1. The *Direct Linear Interpolation* method is used almost exclusively to generate animations in currently existing systems. Including the results for this method gives us an estimate on how much the currently discussed method could improve existing systems.

Scenario 1

Figure 4.11 shows the result of applying the *Orthogonal Interpolation* animation method to scenario 1. The numerical results of the animation in regard to our measures are summarized in the following

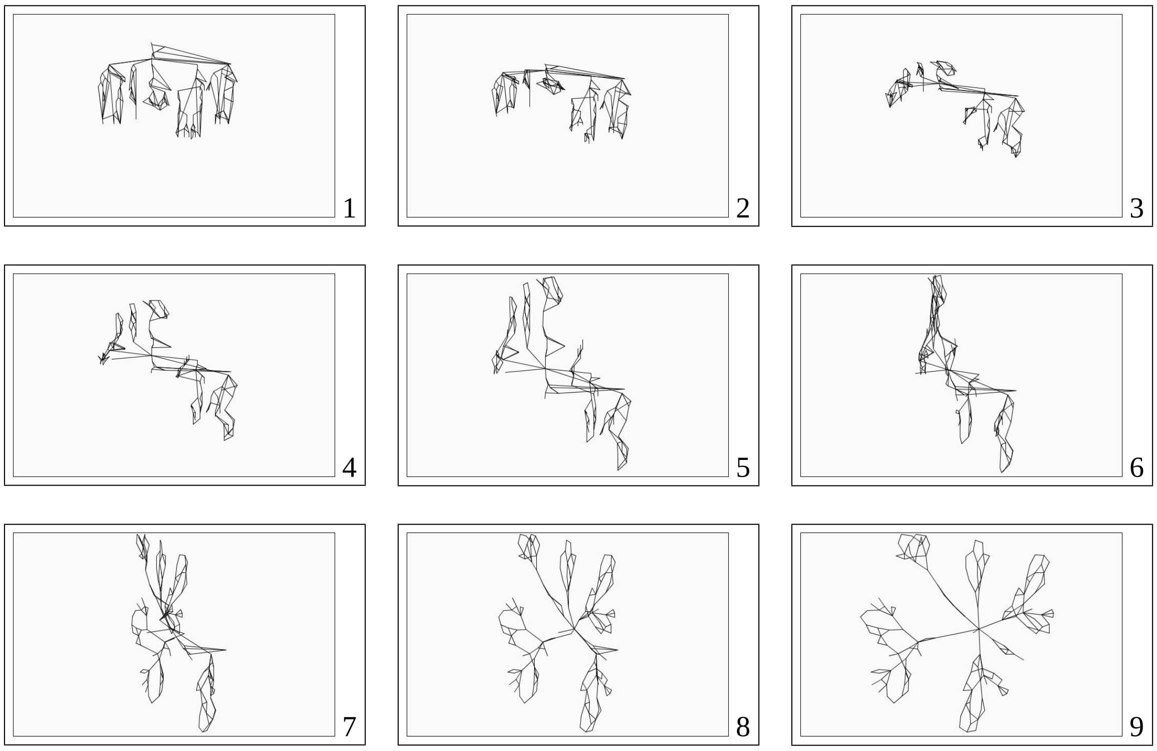
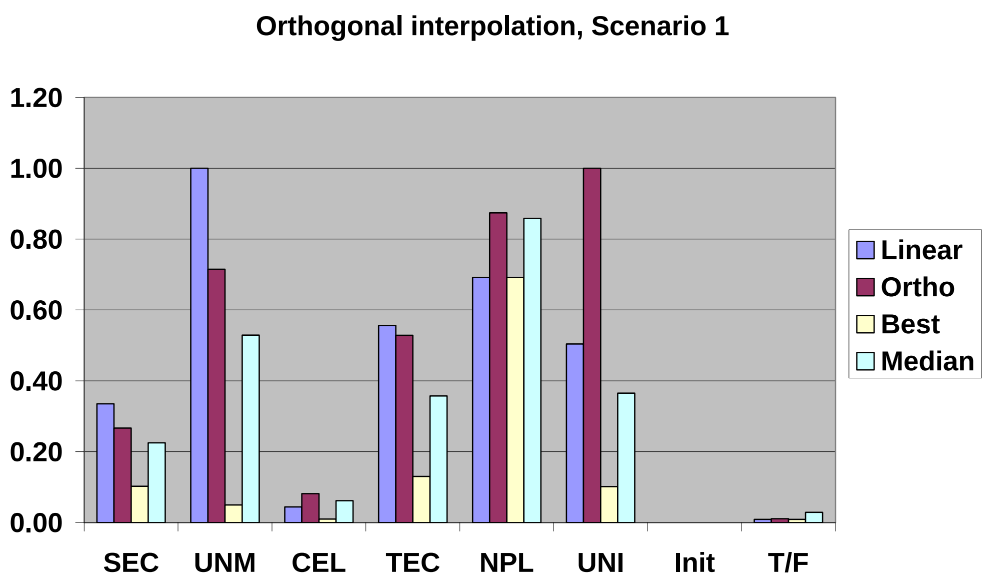


Figure 4.11: This figure shows 9 snapshots of a 101 frame animation. The animation is the result of applying the *Orthogonal Interpolation* animation method to scenario 1.

table:

Method	SEC	UNM	CEL	TEC	NPL	UNI	Init	T/F
Ortho	103	3.589E+09	3,418,826	1,990	439,772	256	0	0.6
Linear	130	5.020E+09	1,843,276	2,095	347,983	129	0	0.5
Best	40	2.484E+08	402,170	490	347,983	26	0	0
Median	87	2.656E+09	2,577,963	1,346	431,802	94	0	1.6



The results are similar to the results for the *Direct Linear Interpolation* method. We notice that the results for *unnecessary node intersections* (UNI) and *constant edge length* (CEL) for the *Orthogonal Interpolation* method are significantly worse than the results for the *Direct Linear Interpolation* method. However, the following sections show that this behavior is specific to this scenario and is not a general trend.

Scenario 1b

Figure 4.12 shows the result of applying the *Orthogonal Interpolation* animation method to scenario 1b. The numerical results of the animation in regard to our measures are summarized in the

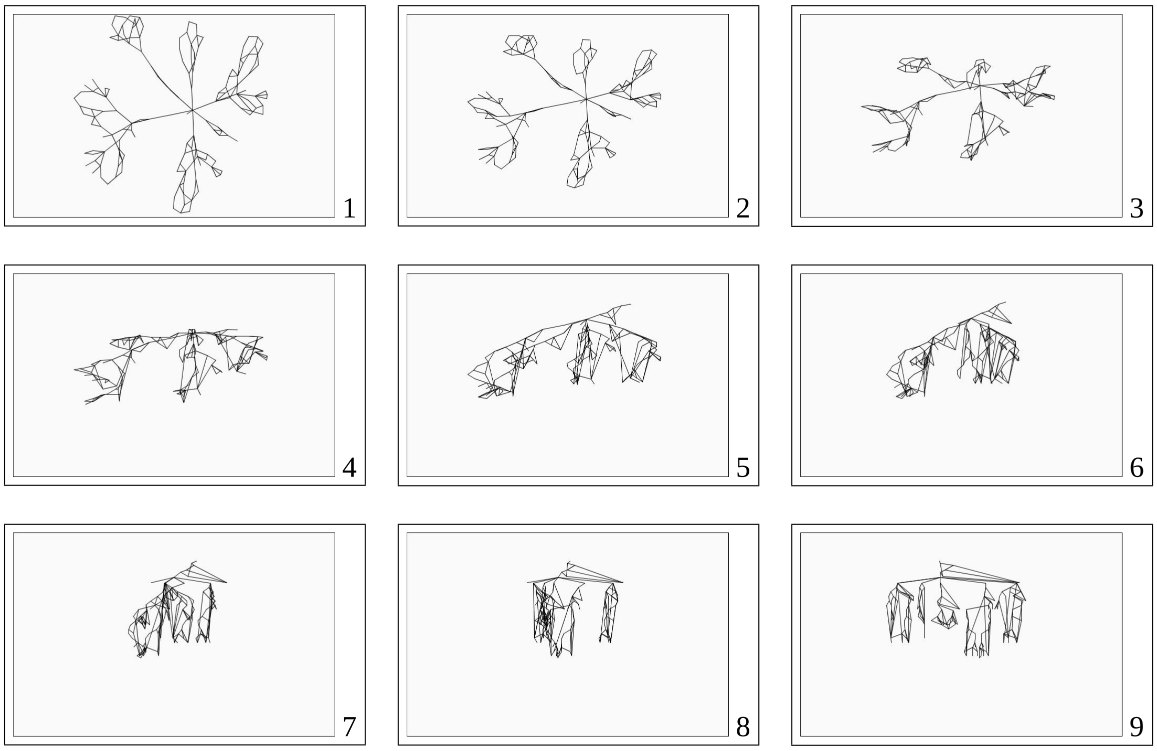
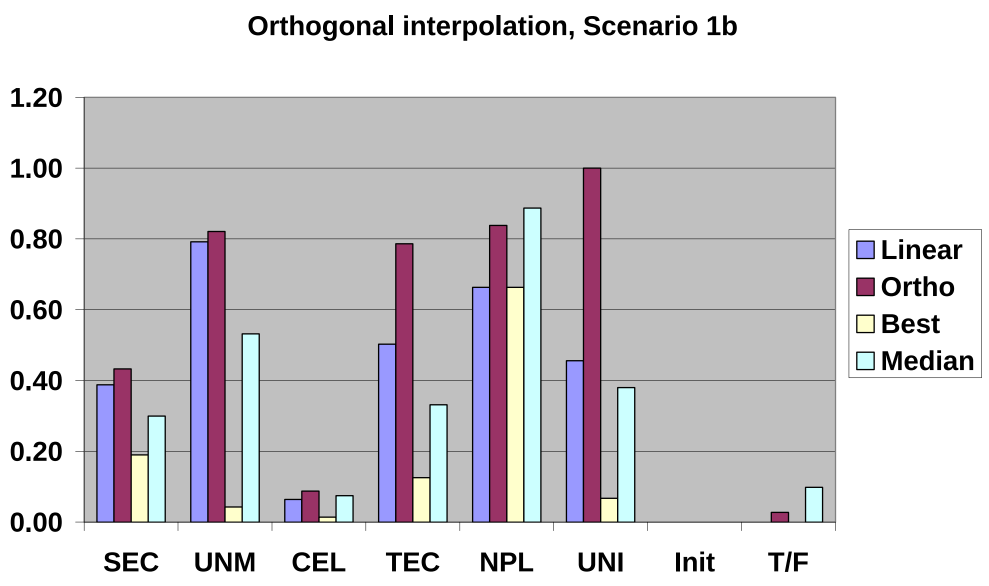


Figure 4.12: This figure shows 9 snapshots of a 101 frame animation. The animation is the result of applying the *Orthogonal Interpolation* animation method to scenario 1b.

following table:

Method	SEC	UNM	CEL	TEC	NPL	UNI	Init	T/F
Ortho	145	5.206E+09	2,519,567	3,345	439,772	283	0	1.4
Linear	130	5.020E+09	1,843,276	2,138	347,983	129	0	0.0
Best	64	2.697E+08	395,880	533	347,983	19	0	0
Median	100	3.371E+09	2,146,880	1,409	465,534	108	0	5.1



The nodes are always moved along the y-axis first and then along the x-axis. This means that the animation is generally not symmetric for exchanged initial and target positions. It is therefore not surprising that the results for scenario 1b are not identical to the results for scenario 1.

Scenario 2

Figure 4.13 shows the result of applying the *Orthogonal Interpolation* animation method to scenario 2. The numerical results of the animation in regard to our measures are summarized in the following

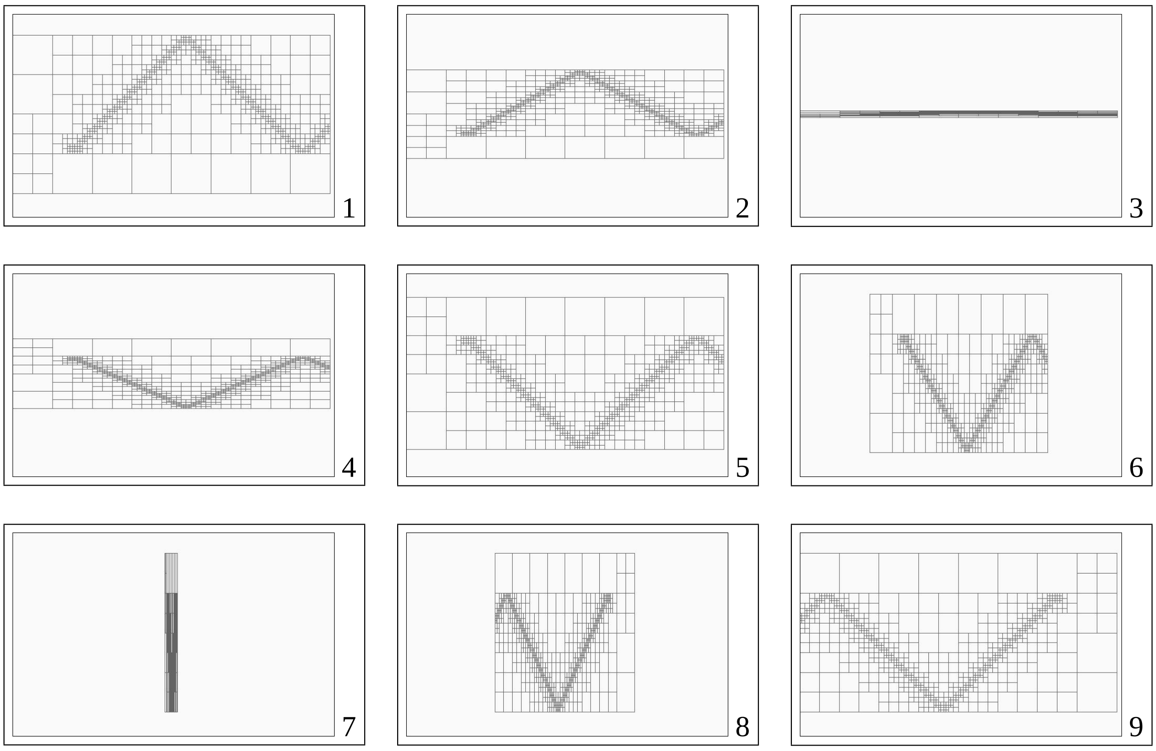
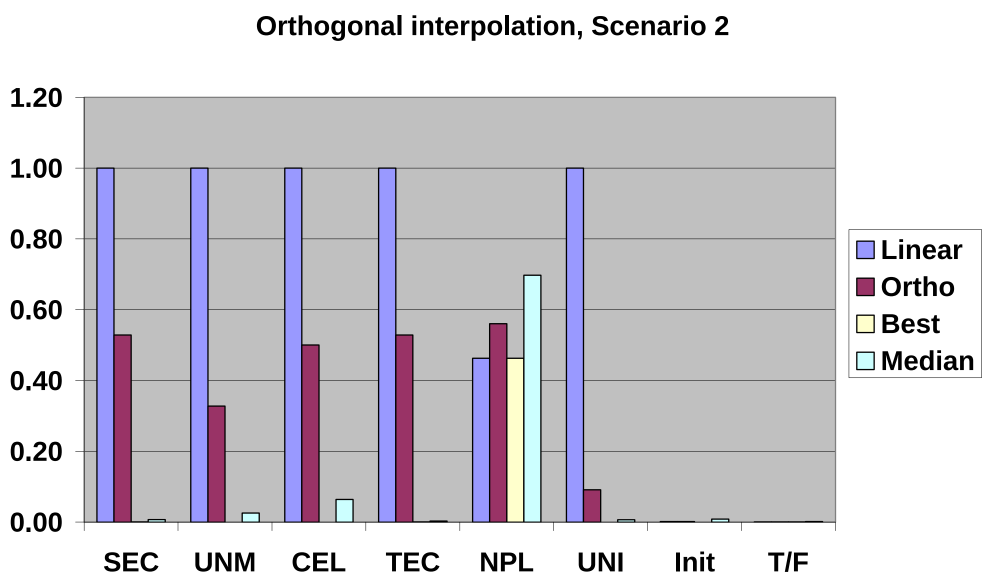


Figure 4.13: This figure shows 9 snapshots of a 101 frame animation. The animation is the result of applying the *Orthogonal Interpolation* method to scenario 2.

table:

Method	SEC	UNM	CEL	TEC	NPL	UNI	Init	T/F
Ortho	6,811	5.489E+09	106,560	687,955	532,660	347,943	50	1.6
Linear	12,891	1.675E+10	212,992	1,301,941	440,110	3,813,032	50	1.6
Best	4	1.066E-20	0	374	440,110	0	0	1
Median	92	4.259E+08	13,608	3,468	662,794	24,579	270	4.7



Although the results for *Orthogonal Interpolation* method are significantly better than the results for the *Direct Linear Interpolation* method, they are still far from the optimum in this scenario.

Scenario 2b

Figure 4.14 shows the result of applying the *Orthogonal Interpolation* method to scenario 2b. The

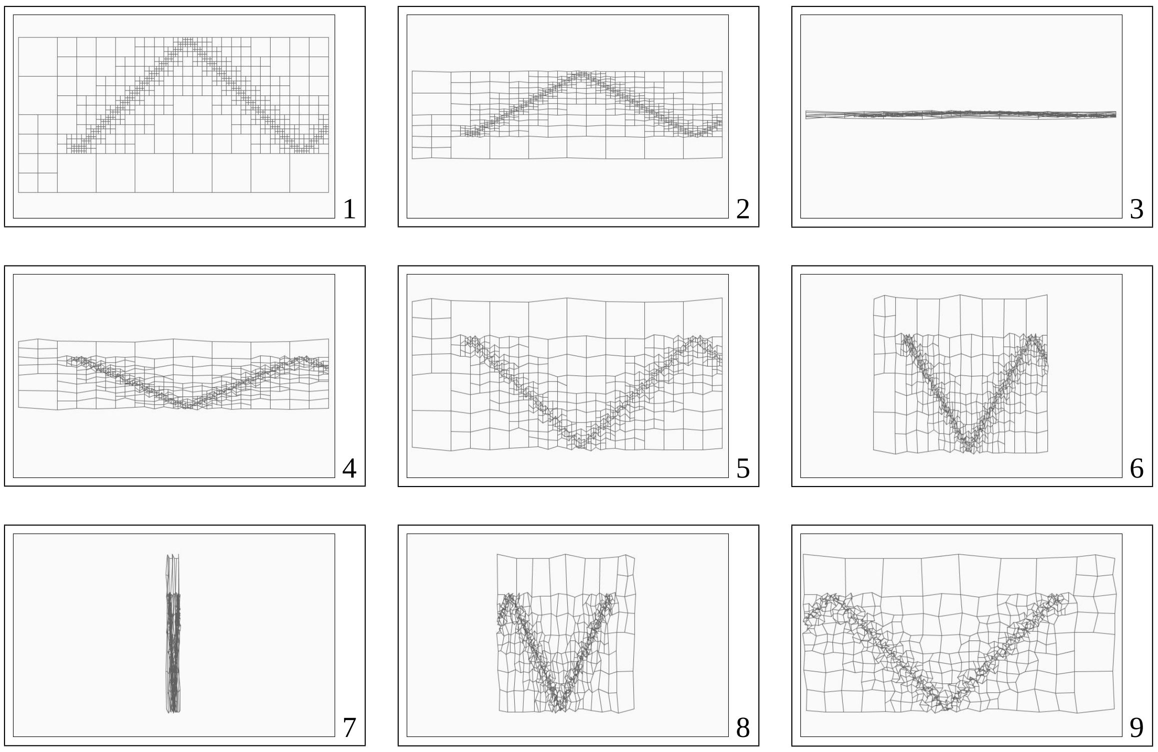
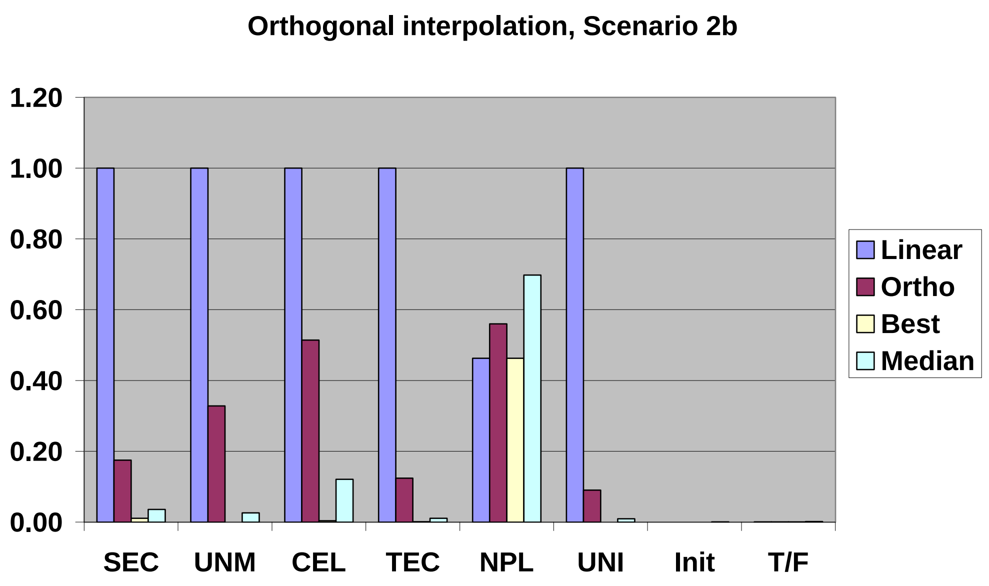


Figure 4.14: This figure shows 9 snapshots of a 101 frame animation. The animation is the result of applying the *Orthogonal Interpolation* method to scenario 2b.

numerical results of the animation in regard to our measures are summarized in the following table:

Method	SEC	UNM	CEL	TEC	NPL	UNI	Init	T/F
Ortho	1,695	5.484E+09	103,893	72,066	532,091	322,941	0	2.1
Linear	9,681	1.670E+10	202,046	582,018	439,723	3,577,834	0	1.0
Best	105	6.805E+03	750	612	439,723	749	0	1
Median	347	4.339E+08	24,396	6,301	662,911	33,607	30	4.3



The results for scenario 2b are consistent with the remarks for the previous scenarios.

Scenario 3

Figure 4.15 shows the result of applying the *Orthogonal Interpolation* method to scenario 3. The

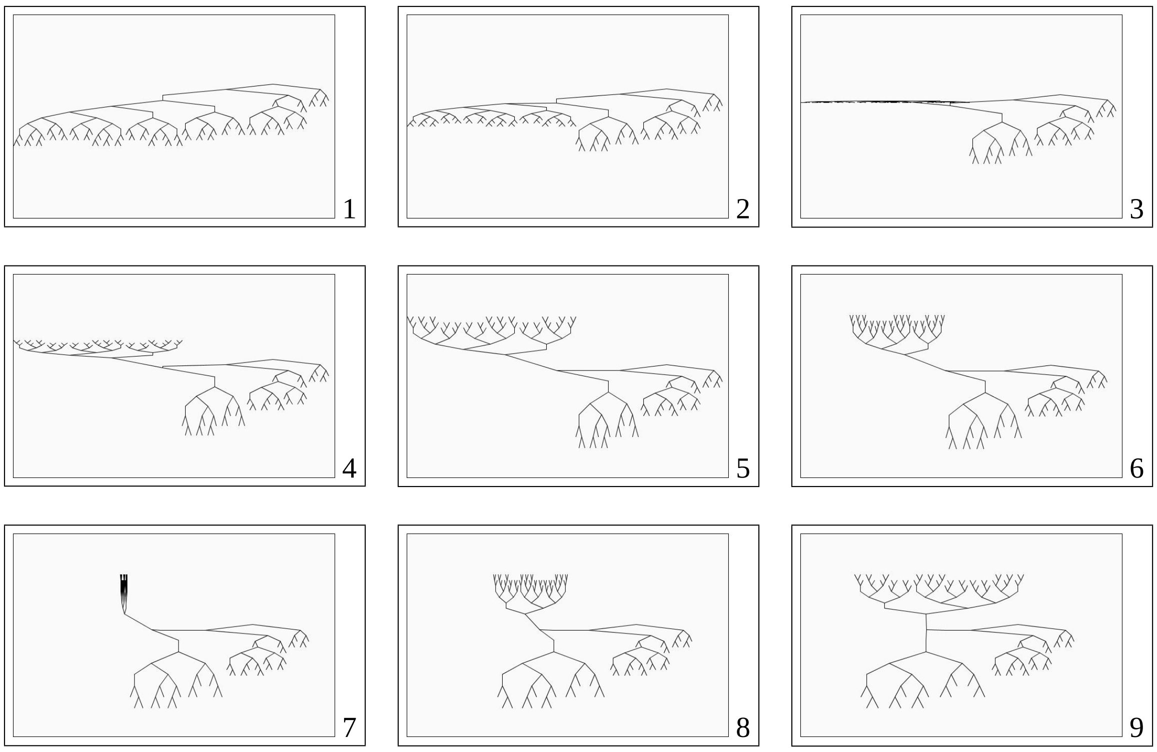
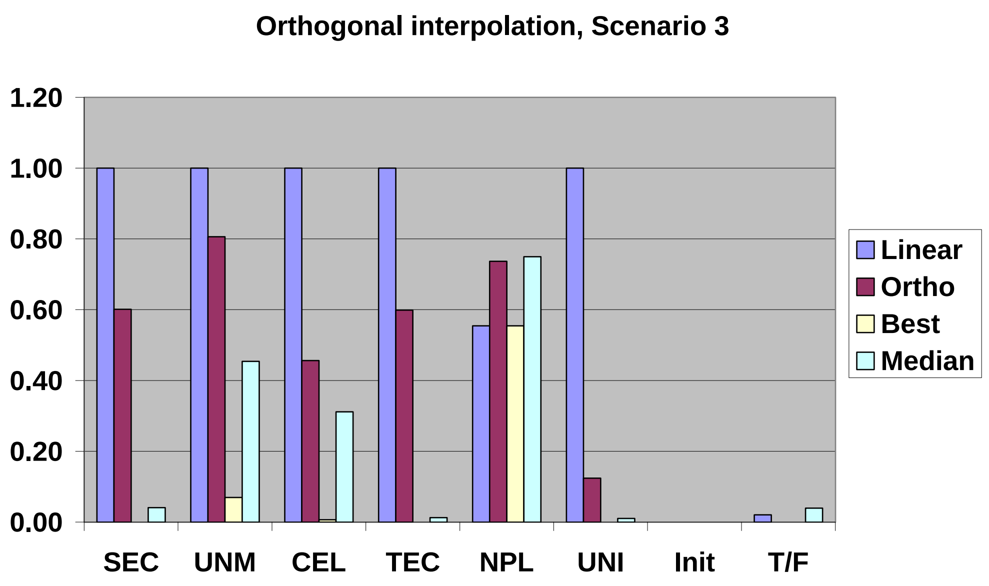


Figure 4.15: This figure shows 9 snapshots of a 101 frame animation. The animation is the result of applying the *Orthogonal Interpolation* method to scenario 3.

numerical results of the animation in regard to our measures are summarized in the following table:

Method	SEC	UNM	CEL	TEC	NPL	UNI	Init	T/F
Ortho	24	3.194E+08	52,429	2,428	125,880	2,916	0	0.0
Linear	40	3.962E+08	114,892	4,055	94,772	23,519	0	0.6
Best	0	2.753E+07	720	0	94,772	0	0	0
Median	2	1.799E+08	35,772	51	128,102	245	0	1.1



The results for scenario 3 are consistent with the remarks for the previous scenarios.

Scenario 4

Figure 4.16 shows the result of applying the *Orthogonal Interpolation* method to scenario 4. The

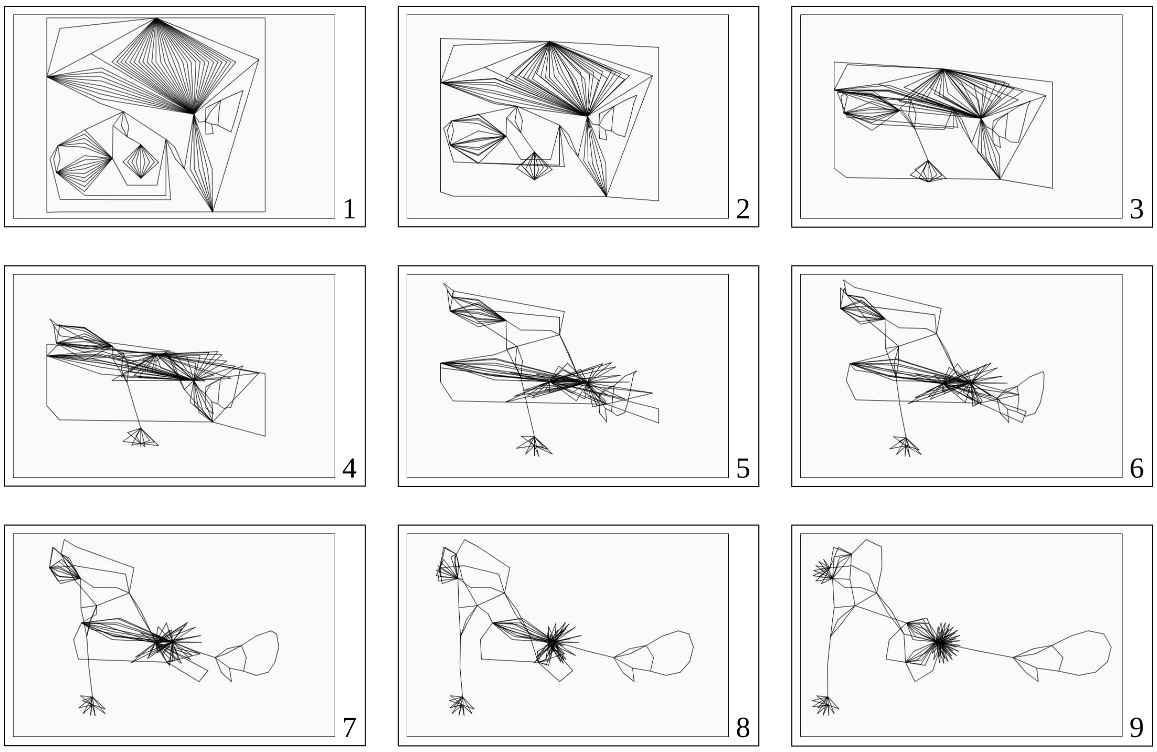
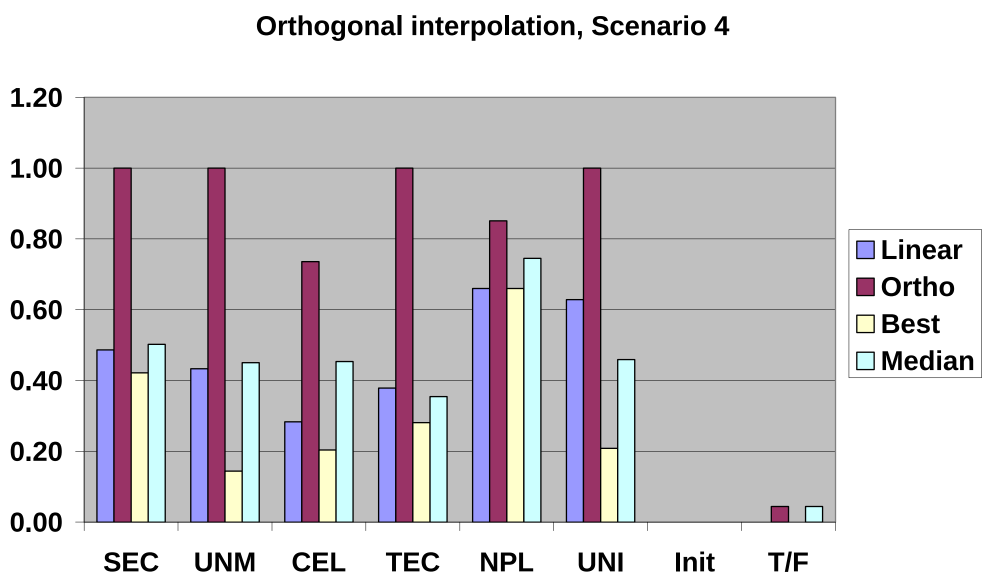


Figure 4.16: This figure shows 9 snapshots of a 101 frame animation. The animation is the result of applying the *Orthogonal Interpolation* method to scenario 4.

numerical results of the animation in regard to our measures are summarized in the following table:

Method	SEC	UNM	CEL	TEC	NPL	UNI	Init	T/F
Ortho	389	6.645E+08	3,446,831	3,509	155,495	269	0	0.5
Linear	190	2.877E+08	1,326,461	1,328	120,561	169	0	0.0
Best	164	9.562E+07	954,237	985	120,561	56	0	0
Median	196	2.991E+08	2,125,756	1,244	136,133	124	0	0.5



The results of the *Orthogonal Interpolation* method for scenario 4 are significantly worse than the results of most other methods, including the *Direct Linear Interpolation* method. The animation itself, however, exhibits the same characteristic pseudo three-dimensional rotation as we have seen in many other scenarios and looks “aesthetically pleasing”. The influence of the illusion of three-dimensional movement seems to be strong, but it cannot be measured by our criteria.

Scenario 5

Figure 4.17 shows the result of applying the *Orthogonal Interpolation* method to scenario 5. The

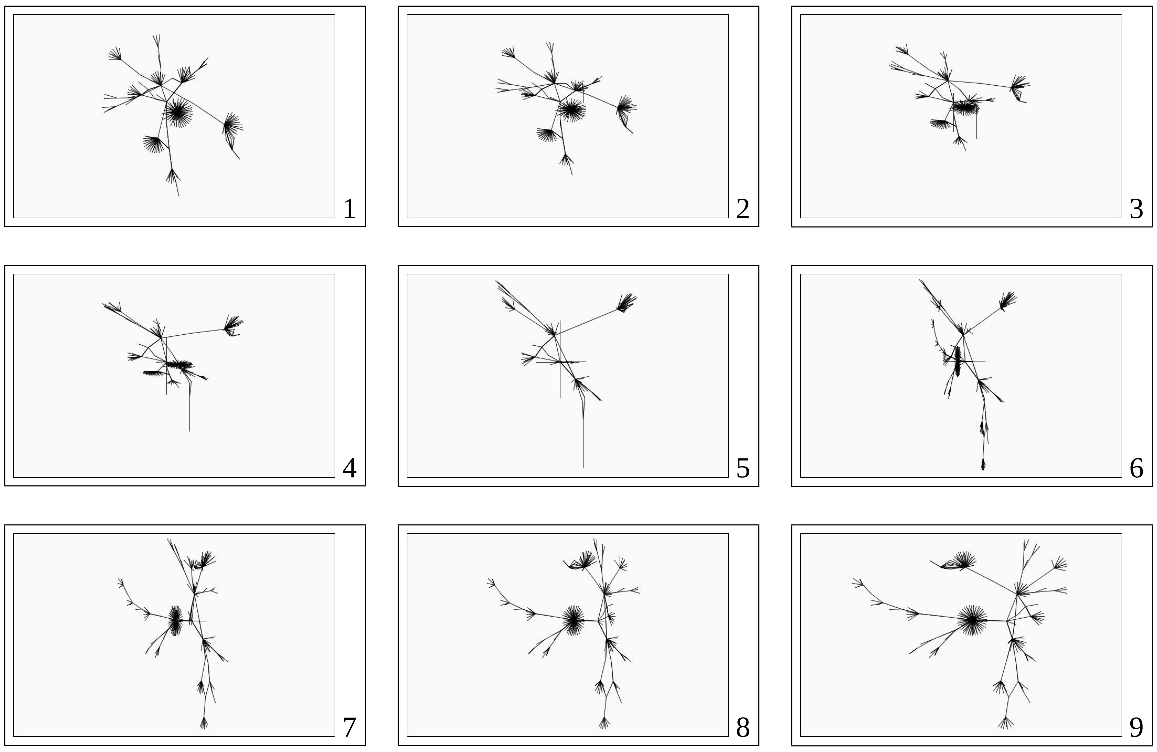
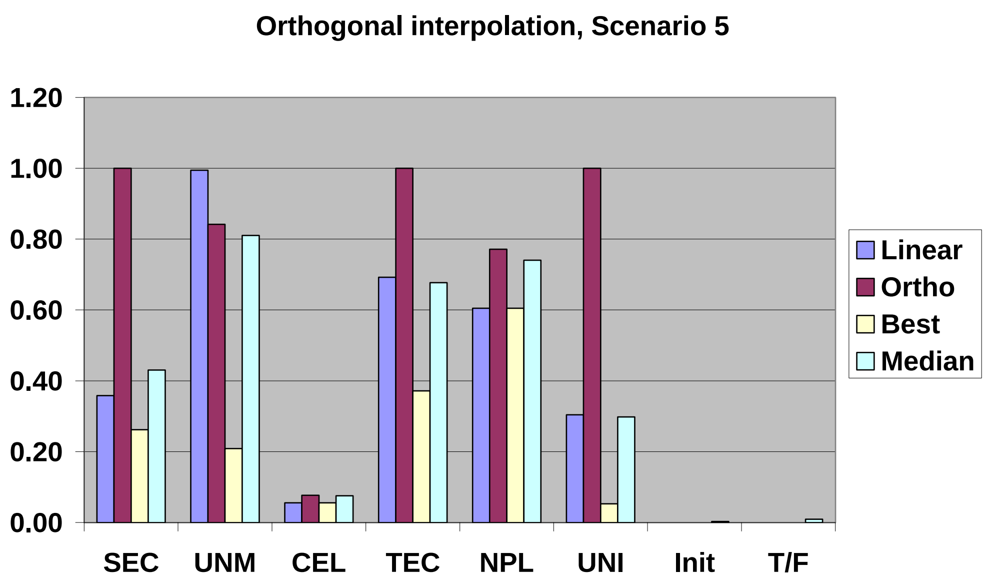


Figure 4.17: This figure shows 9 snapshots of a 101 frame animation. The animation is the result of applying the *Orthogonal Interpolation* method to scenario 5.

numerical results of the animation in regard to our measures are summarized in the following table:

Method	SEC	UNM	CEL	TEC	NPL	UNI	Init	T/F
Ortho	1,174	1.245E+10	3,309,387	22,711	761,707	911	0	0.0
Linear	420	1.470E+10	2,399,401	15,725	597,271	277	0	0.0
Best	308	3.088E+09	2,399,401	8,438	597,271	48	0	0
Median	505	1.198E+10	3,262,048	15,373	731,150	272	55	1.1



We notice significantly bad values for *static edge crossings* (SEC) and *unnecessary node intersections* (UNI). Again, we experience the illusion of a three-dimensional rotation of the graph. The animation itself is not convincing however. The story behind the changes to the graph, that is the expanding and collapsing of clusters of nodes does not become apparent during the animation.

Scenario 6

Figure 4.18 shows the result of applying the *Orthogonal Interpolation* method to scenario 6. The

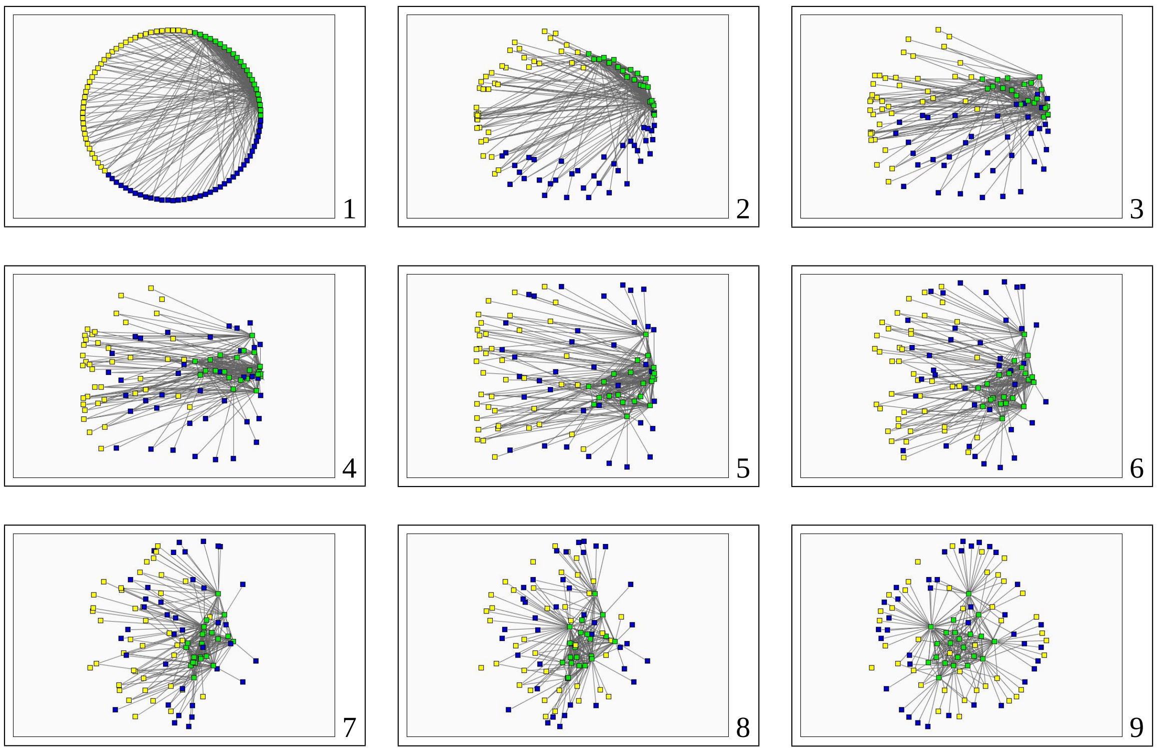
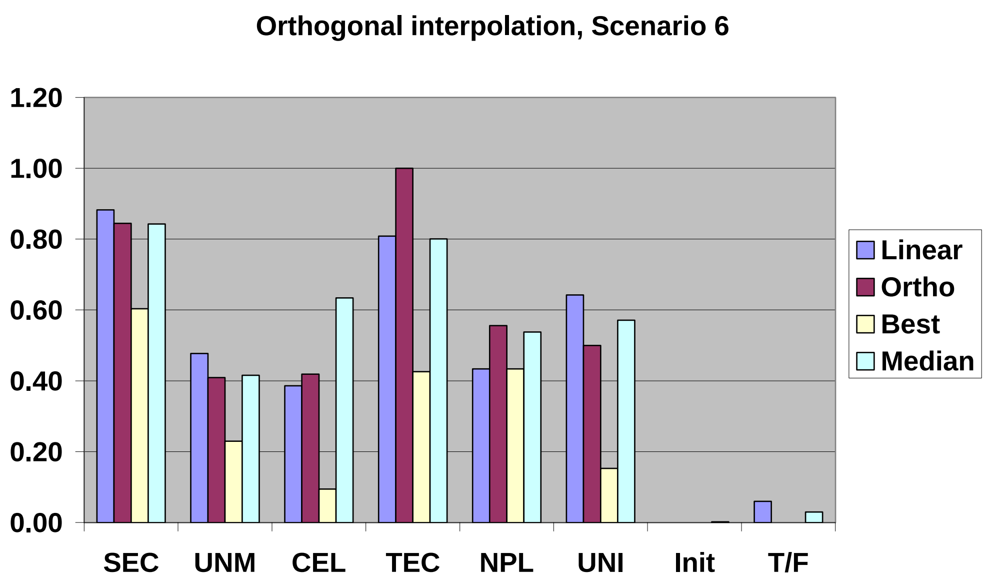


Figure 4.18: This figure shows 9 snapshots of a 101 frame animation. The animation is the result of applying the *Orthogonal Interpolation* method to scenario 6.

numerical results of the animation in regard to our measures are summarized in the following table:

Method	SEC	UNM	CEL	TEC	NPL	UNI	Init	T/F
Ortho	6,788	2.044E+08	4,528,321	19,494	94,808	144	0	0.0
Linear	7,095	2.385E+08	4,174,533	15,762	73,957	185	0	0.5
Best	4,853	1.147E+08	1,020,039	8,301	73,957	44	0	0
Median	6,777	2.077E+08	6,856,478	15,610	91,672	165	25	0.2



The results for scenario 6 are consistent with the remarks for the previous scenarios.

Scenario 7

Figure 4.19 shows the result of applying the *Orthogonal Interpolation* method to scenario 7. The

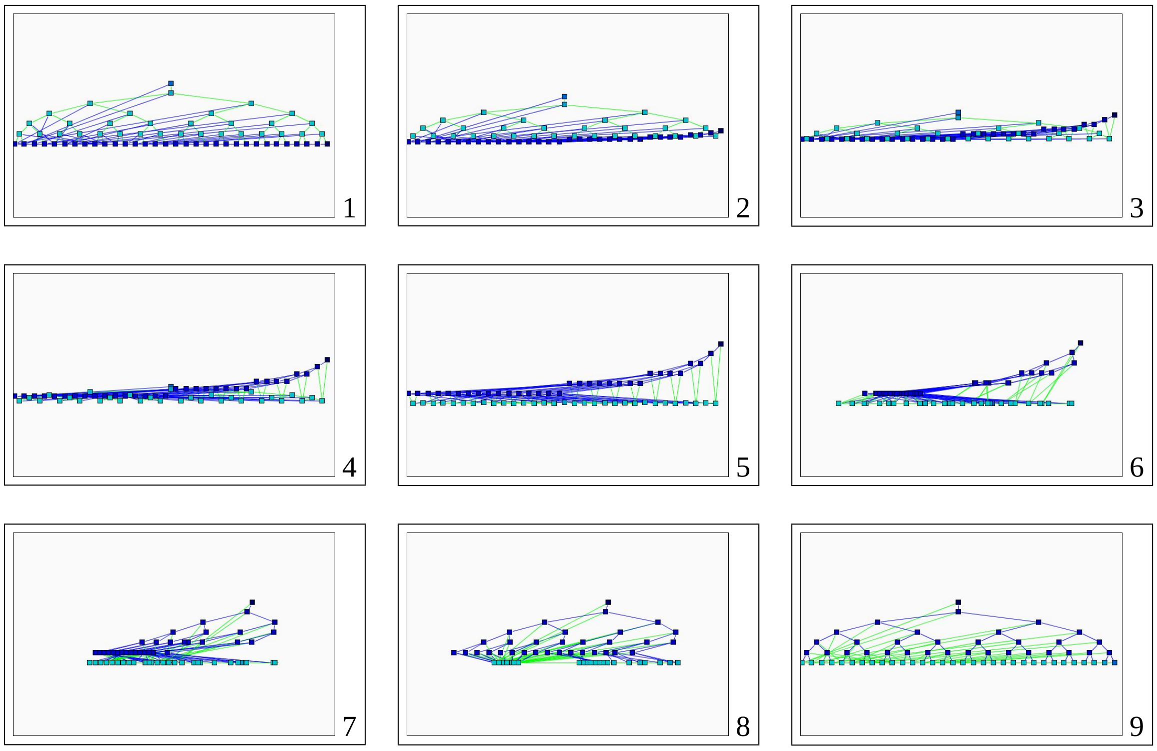
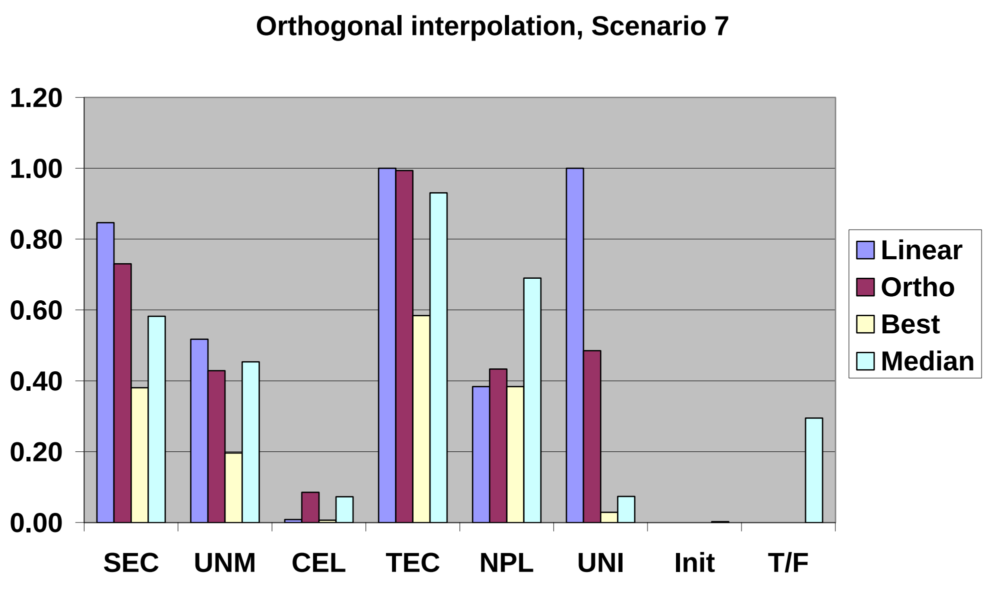


Figure 4.19: This figure shows 9 snapshots of a 101 frame animation. The animation is the result of applying the *Orthogonal Interpolation* method to scenario 7.

numerical results of the animation in regard to our measures are summarized in the following table:

Method	SEC	UNM	CEL	TEC	NPL	UNI	Init	T/F
Ortho	873	4.028E+07	953,805	4,942	28,418	1,305	0	0.0
Linear	1,011	4.858E+07	90,948	4,975	25,170	2,692	0	0.0
Best	454	1.841E+07	71,857	2,905	25,170	77	0	0
Median	696	4.262E+07	816,052	4,631	45,283	198	25	1.1



The results for scenario 7 are consistent with the remarks for the previous scenarios.

Scenario 8

Figure 4.20 shows the result of applying the *Orthogonal Interpolation* method to scenario 8. The

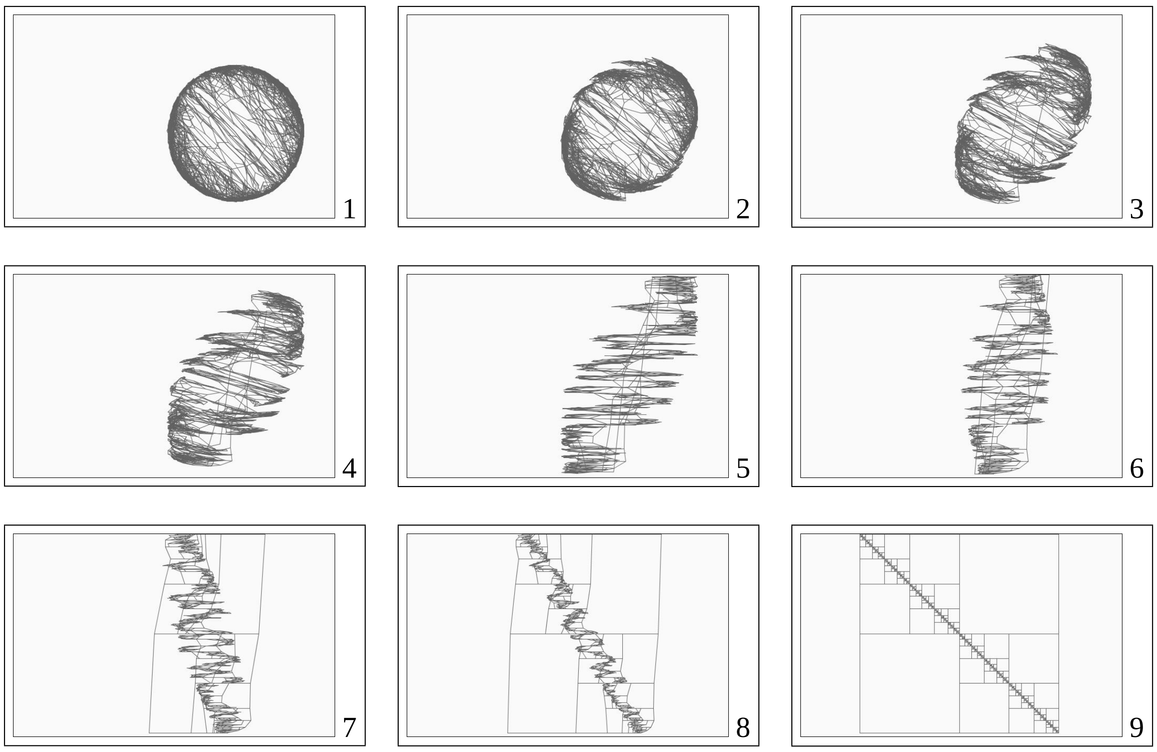
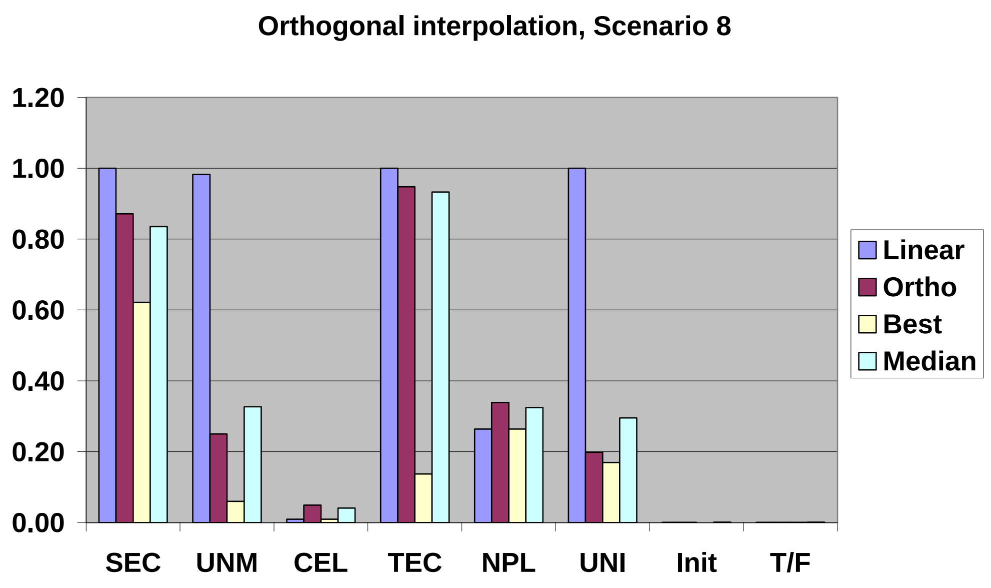


Figure 4.20: This figure shows 9 snapshots of a 101 frame animation. The animation is the result of applying the *Orthogonal Interpolation* method to scenario 8.

numerical results of the animation in regard to our measures are summarized in the following table:

Method	SEC	UNM	CEL	TEC	NPL	UNI	Init	T/F
Ortho	40,254	7.195E+08	22,157	2,002,272	372,986	1,470,055	50	4.2
Linear	46,185	2.829E+09	4,138	2,112,739	290,510	7,427,591	50	7.2
Best	28,706	1.718E+08	4,138	289,169	290,510	1,256,611	0	4
Median	38,574	9.407E+08	18,486	1,971,003	357,359	2,192,830	80	13.7



The results for scenario 8 are consistent with the remarks for the previous scenarios.

4.2.4 Summary

In most cases the orthogonal interpolation method produces "aesthetically pleasing" animations. This is mainly due to the fact that the human brain often interprets the resulting motion of the graph as a three-dimensional rotation of a rigid object in space.

As we have seen in scenario 4, this optical illusion can dominate all other criteria for good animations. That is, although the measures are exceptionally bad, the animation shows the characteristic three-dimensional illusion and is thus perceived as aesthetically pleasing.

Scenario 5, however, suggests that the fact that an animation is aesthetically pleasing does not necessarily imply that it is also well suited for communicating the changes to a graph or helping users adapt their mental map. Although the animation for scenario 5 shows the typical illusion and looks aesthetically pleasing it is very difficult to understand what is happening to the graph. The fact that clusters of nodes are expanded and contracted does not become obvious at all.

Orthogonal interpolation produces "good looking" animations in the majority of cases. Its use as a special effect in the user interface of a graph drawing system can therefore be quite rewarding.

We have seen, however, that this does not imply that orthogonal interpolation also achieves the goal of helping users in adapting the mental map. The formal measures may be a better guide to usefulness than the beauty of an animation.

CHAPTER 5

Force Simulation Methods

The force directed paradigm tries to find a solution to a minimization problem by encoding the relevant criteria as forces in a physical system. By simulating this system a solution to the minimization problem can be found. Although it can generally not be guaranteed that the global minimum is found, force directed approaches have shown good results in many applications. In this chapter we develop models to formulate the graph animation problem as a physical system and compute animations by simulating these models. Force directed approaches are especially suitable for animations as node positions are modified continuously, implicitly defining possible frames for an animation by the iterations of the algorithm.

5.1 Force Directed Method

5.1.1 The Method

The well known and very popular spring embedder layout paradigm for graphs can be intuitively adapted to compute graph animations. The traditional force directed layout algorithm applies two kinds of forces to each node in order to compute a layout:

- Nodes repel each other with a force proportional to their distance.
- Nodes which are connected by an edge are repelled or attracted depending on whether they are closer or further apart than the optimal edge length.

If the target layout is identical to the layout computed by the spring embedder algorithm then an animation can be easily created by just displaying the graph after each iteration of node updates. We can modify this approach to compute animations between arbitrary graph drawings. Given a graph $G = (E, V)$ and an initial frame F_0, as well as a target frame F_1, $F_0, F_1 : V \to \mathbb{R}^2$, we can compute an animation between arbitrarily given start and end positions by replacing the traditional forces with the following ones:

1. Nodes are attracted to each other or repelled from each other depending on whether their distance is smaller or bigger than the linearly interpolated distance between their positions in the initial and target frame. Given a function $P_v : [0 \dots 1] \to \mathbb{R}^2$ specifying the position of a node $v \in V$ at time t, $1_{(v,u)}(t)$ being the unit vector parallel to the vector $P_u(t) - P_v(t)$, and the distance function $d_{(u,v)}(t) : [0 \dots 1] \to \mathbb{R}$ determining the distance between two nodes $u, v \in V$ at time t of the animation, the nodes of the graph exert the following forces on each node $v \in V$ at time t:

$$InterNodeForce_v(t) = \sum_{u \in (V \backslash v)} 1_{(v,u)}(t) \left(d_{(u,v)}(t) - \left((1-t)\, d_{(u,v)}(0) + t d_{(u,v)}(1) \right) \right)$$

2. Nodes are attracted by their target positions. The force of the attraction increases with the duration of the animation. Given the current positions $F_t : V \to \mathbb{R}^2$ and the distance function $d : (V, V) \to \mathbb{R}$, this exerts the following force on a node v at time t:

$$TargetAttractionForce_v(t) = P_v(1) - P_v(t)$$

Thus, given a weight function $T : [0 \dots 1] \to [0 \dots 1]$ with $T(0) = 0$ and $T(1) = 1$, the force on each node $v \in V$ at time t is given by:

$$Force_v(t) = (1 - T(t))\, InterNodeForce_v(t) + T(t)\, TargetAttractionForce_v(t)$$

Newton's second law of motion states that force is equal to mass times acceleration: $F = ma$. Assuming a mass of 1 we can use this law to compute the acceleration, speed and finally the position for each node. The position of each node at time t is hence given by the following set of differential equations:

$$P_v(t) = \int \int Force_v(t)\, dt, \quad v \in V$$

This path definition computes node paths according to the simulated physical system. Unfortunately it does not guarantee that the nodes are actually at their designated target positions at time $t = 1$.

This is essential for a graph animation however. By simplifying the physical system and directly interpreting the force applied to a node as its speed[1], we can guarantee that all nodes reach their target positions in time. We thus use the following path definition:

$$P_v\left(t\right) = \int Force_v\left(t\right)dt, \;\; v \in V$$

While inter-node relations dominate the simulation in the beginning the nodes become more and more attracted by their target positions during the animation. At $t = 1$ the inter-node forces become 0 and the nodes move directly to their target position. It is thus guaranteed that the animation terminates with all nodes at their target position.

The set of differential equations $P_v\left(t\right)$ cannot be solved analytically. It can, however, easily be approximated using standard numerical iteration methods. A problem arises when two nodes lie at the same position. Although the resulting node and edge forces between these nodes are finite in this case, the direction of the force is not specified. As the repulsive force between nodes does not approach infinity for small distances between the nodes this case can actually happen, even in the continuous model. In our algorithm we ignore forces which do not have a direction and rely on the remaining forces to pull the overlying nodes apart. In practice we found this approach to work well without any noticeable impact on the performance of the algorithm.

The influence of the balancing function T on the speed of the animation is not straightforward and very difficult to control in practice. Results for a given balancing function usually differ between different animation problem instances. For the empirical evaluation of the *Force Directed* method in section 5.1.3 we use the identity function for T.

The quality of the resulting animation is also dependent on the quantization. If only few frames are displayed and thus the steps of t are large, the force vectors and therefore the nodes paths are noticeably different to an animation with small steps of t. For the empirical evaluation of the *Force Directed* method in section 5.1.3 we use 101 frames for each scenario.

5.1.2 Analysis

In animations which are generated by the *Force Directed* method, the paths of the nodes are only given indirectly, as the solution of a non-linear set of differential equations. This set of equations is not analytically solvable. A full analysis of most measures, however, would require that we have

[1]This is also known as Aristotle’s “law” of motion

knowledge of the node paths. In this section we therefore make a few analytical remarks, without giving a full analysis.

1. Init

   All necessary computations are done during the animation. No preprocessing is necessary.

2. T/F

   The computational cost for each frame is very high. As each node exerts a force on each other node $O\left(n^2\right)$ operation have to be performed for each frame. This limits the application of this method to small graphs.

3. UNI

   The number of *unnecessary node intersections* is directly addressed by the *Force Directed* method through the inter-node forces. The tendency to avoid node intersections can be adjusted by changing the weight of the inter-node forces with respect to the target attraction forces. Intuition tempts us to assume that we can avoid node intersections completely by using a high enough weight for the node repulsion forces. Unfortunately, intuition fails in this case. Let us consider the pathological scenario illustrated in figure 5.1. The graph consists of

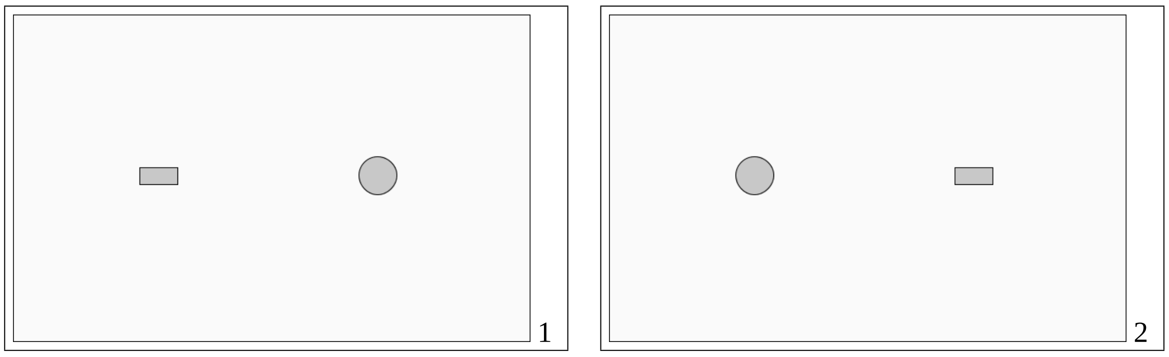


Figure 5.1: This scenario illustrates that node intersection can generally not be avoided by high weights for inter node forces. As the target attraction forces directly oppose the node repulsion forces in this scenario, the balancing function guarantees that the target attraction forces cancel the node repulsion forces at some stage during the animation. The nodes, driven exclusively by the target attraction forces at this stage, then intersect on their way to their target positions

   only two nodes which swap their position from the initial drawing to the target drawing. The target attraction forces are directly opposing the node attraction forces in this scenario. As the node repulsion forces are finite, the balancing function guarantees that that at some stage the target attraction forces become stronger than the node repulsion forces and the two nodes intersect.

4. UNM

   The value of the *uniform node movement* measure is directly addressed by the *Force Directed* method. The inter-node forces try to maintain uniform node distances while the target attraction forces move the nodes to their target positions. The balancing function increases the weight of the target attraction forces in respect to the inter-node forces during the animation. This means that *uniform node movement* can become worse towards the end of the animation if the *Force Directed* method was not able to identify a way in which the nodes can be uniformly moved from their initial positions to their target positions.

5. CEL

   The *constant edge length* measure is closely related to the *uniform node movement* measure. In contrast to the *uniform node movement* measure, however, the *constant edge length* measure only considers pairs of nodes which are connected by an edge. This seems to imply that, indirectly, the *Force Directed* method also achieves good values for *constant edge length*. In many specific cases, however, this indirect support is not strong enough to guarantee good values for *constant edge length*: We found an essential trade-off between *constant edge length* and *uniform node movement* in many scenarios. Further, in most graphs which are based on real world applications, the number of edges is linear in the number of nodes in the graph. For the *Force Directed* animation method, this means that a linear number of forces supporting *constant edge length* is opposed by a quadratic number of forces supporting *uniform node movement*. In these cases *Force Directed* animations achieve comparatively bad results for *constant edge length*.

6. Smoothness

   The motion is the result of simulating a physical system. The resulting motion is therefore smooth if an appropriate force simulation method is used.

7. NPL

   Generally, the internode forces prevent the nodes from travelling on the shortest path to optimize other criteria. Target attraction forces prevent the nodes from assuming arbitrary long paths however.

5.1.3 Empirical Evaluation

Scenario 1

Figure 5.2 shows the result of applying the *Force Directed* method to scenario 1. The numerical

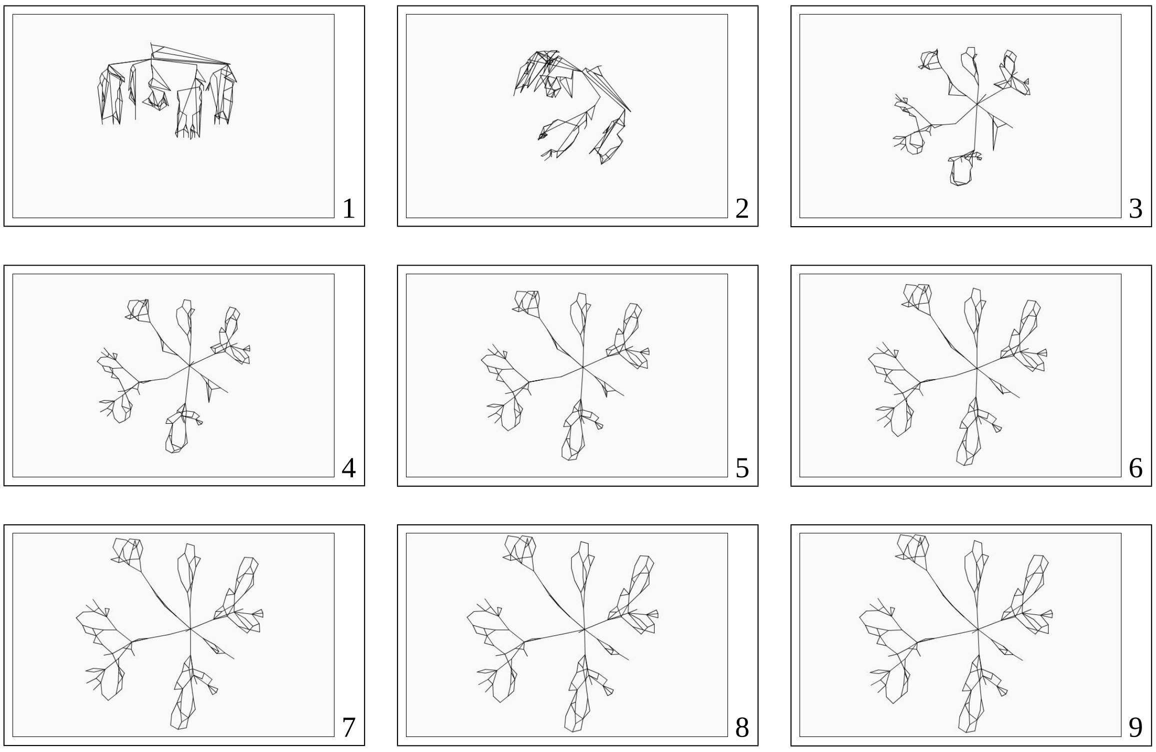
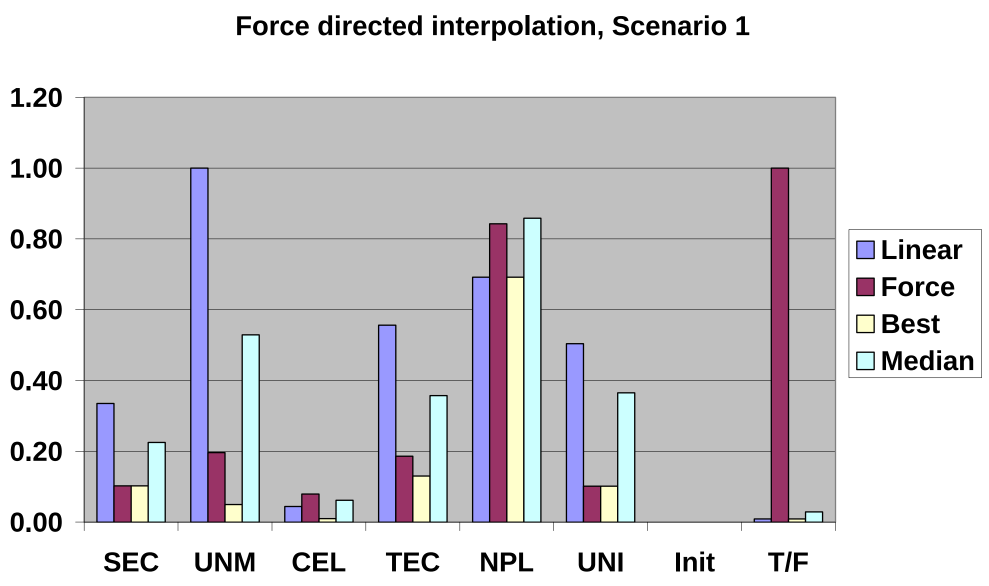


Figure 5.2: This figure shows 9 snapshots of a 101 frame animation. The animation is the result of applying the *Force Directed* method to scenario 1.

results of the animation in regard to our measures are summarized in the following table:

Method	SEC	UNM	CEL	TEC	NPL	UNI	Init	T/F
Force	40	9.840E+08	3,312,649	701	423,831	26	0	55.0
Linear	130	5.020E+09	1,843,276	2,095	347,983	129	0	0.5
Best	40	2.484E+08	402,170	490	347,983	26	0	0
Median	87	2.656E+09	2,577,963	1,346	431,802	94	0	1.6



The *Force Directed* method achieves best overall results for *static edge crossings* (SEC), *uniform node movement* (UNM), and *unnecessary node intersections* (UNI) in scenario 1. The good values for *unnecessary node intersections* and *uniform node movement* are not surprising as these criteria are explicitly represented by forces in the simulated system.

We also notice a comparatively bad result for *constant edge length* (CEL). Although this seems counterintuitive at first, as *constant edge length* is equivalent to *uniform node movement* when only applied to a subset of the node-pairs of the graph, we also find this phenomenon in some of the following scenarios. A detailed discussion which explains this phenomenon can be found in the summary section for the *Force Directed* method on page 170.

It is difficult to control the speed of the animation in the *Force Directed* method. We notice that the major part of the movements are completed in only 50% of the animation time in this scenario.

Scenario 1b

Figure 5.3 shows the result of applying the *Force Directed* method to scenario 1b. The numerical

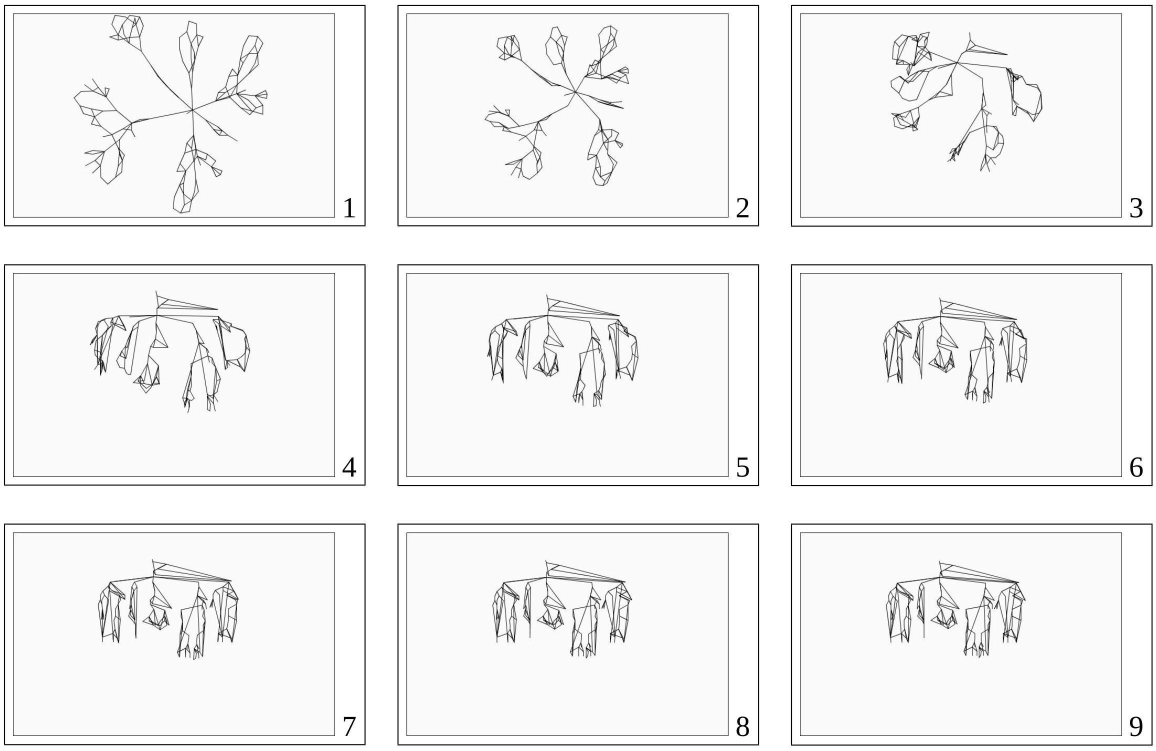
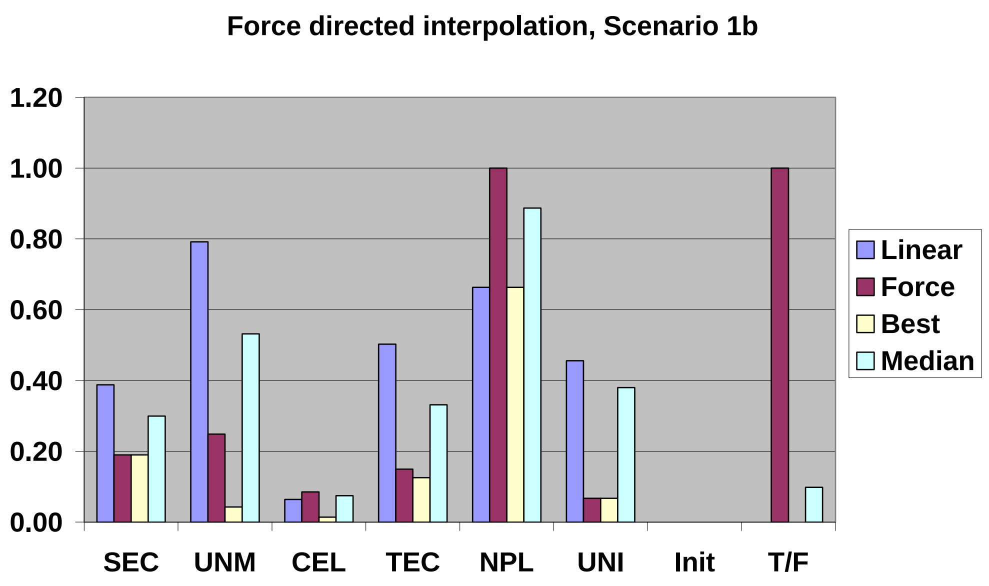


Figure 5.3: This figure shows 9 snapshots of a 101 frame animation. The animation is the result of applying the *Force Directed* method to scenario 1b.

results of the animation in regard to our measures are summarized in the following table:

Method	SEC	UNM	CEL	TEC	NPL	UNI	Init	T/F
Force	64	1.574E+09	2,450,485	635	524,618	19	0	52.2
Linear	130	5.020E+09	1,843,276	2,138	347,983	129	0	0.0
Best	64	2.697E+08	395,880	533	347,983	19	0	0
Median	100	3.371E+09	2,146,880	1,409	465,534	108	0	5.1



The *Force Directed* method generally does not produce symmetric animations for exchanged initial and target drawings. It is therefore not surprising that the results for scenario 1b differ from the results for scenario 1. Nevertheless, the force directed approach again achieves overall best results for *static edge crossings* (SEC), *uniform node movement* (UNM), and *unnecessary node intersections* (UNI). And, again we see comparatively bad results for *constant edge length* (CEL).

Scenario 2

Figure 5.4 shows the result of applying the *Force Directed* method to scenario 2. The numerical

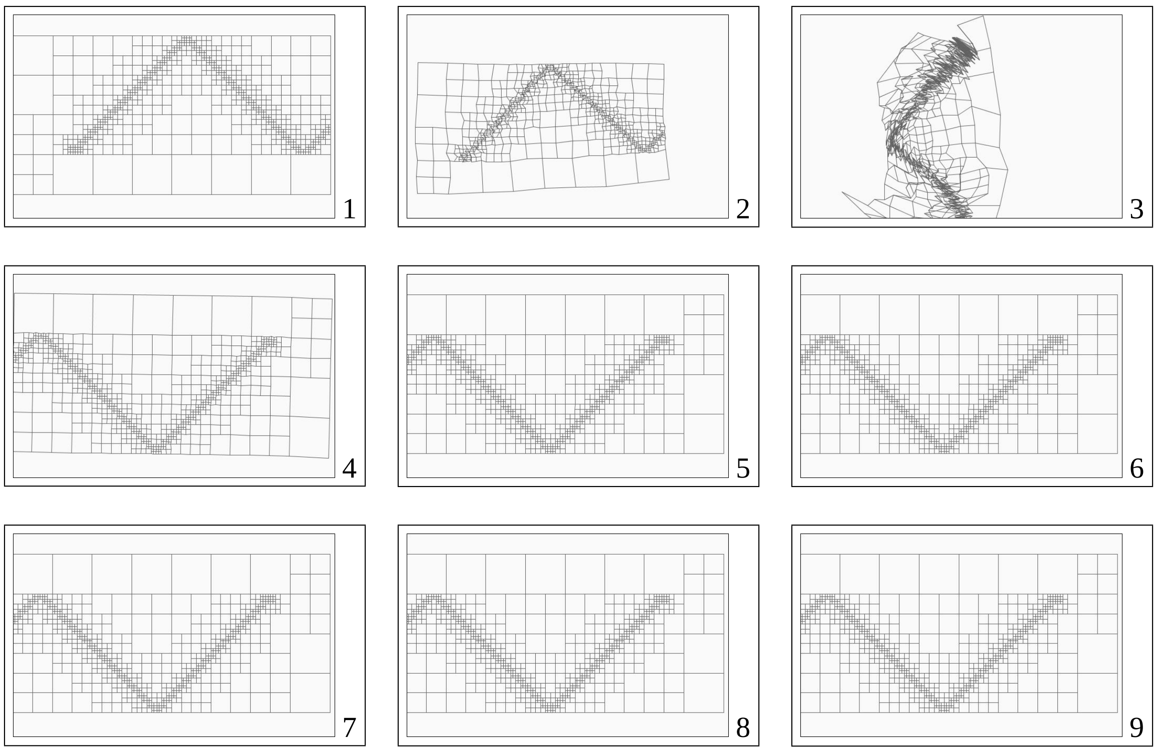
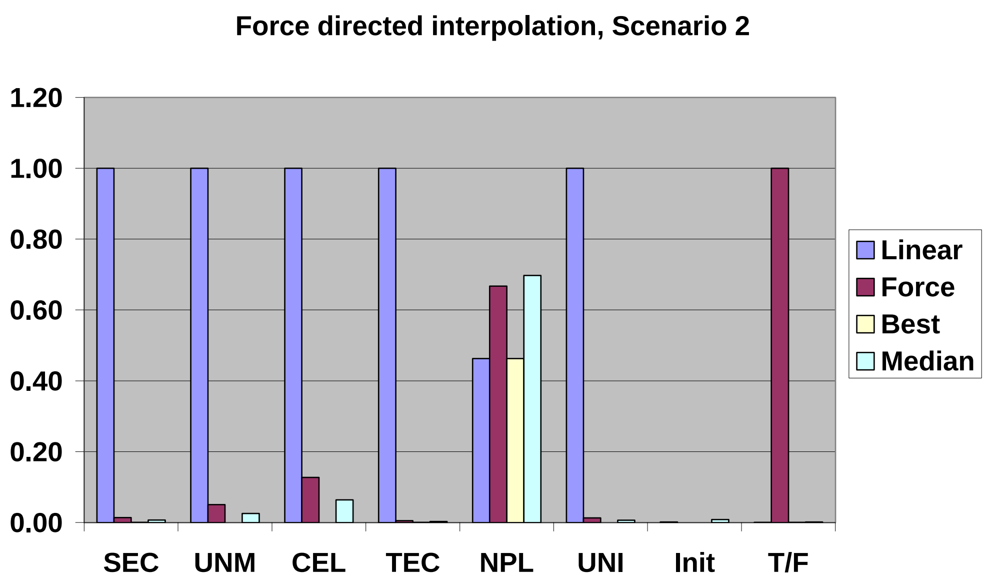


Figure 5.4: This figure shows 9 snapshots of a 101 frame animation. The animation is the result of applying the *Force Directed* method to scenario 2.

results of the animation in regard to our measures are summarized in the following table:

Method	SEC	UNM	CEL	TEC	NPL	UNI	Init	T/F
Force	179	8.419E+08	27,090	6,358	634,338	49,158	0	2,943.1
Linear	12,891	1.675E+10	212,992	1,301,941	440,110	3,813,032	50	1.6
Best	4	1.066E-20	0	374	440,110	0	0	1
Median	92	4.259E+08	13,608	3,468	662,794	24,579	270	4.7



Although the force directed approach fails to find the optimal animation for this scenario (a 180^o rotation of the graph), it succeeds, at least partially, to maintain the structure of the drawing during the animation. The values are significantly better compared to the *Direct Linear Interpolation* method in regard to most measures.

Scenario 2b

Figure 5.5 shows the result of applying the *Force Directed* method to scenario 2b. The numerical

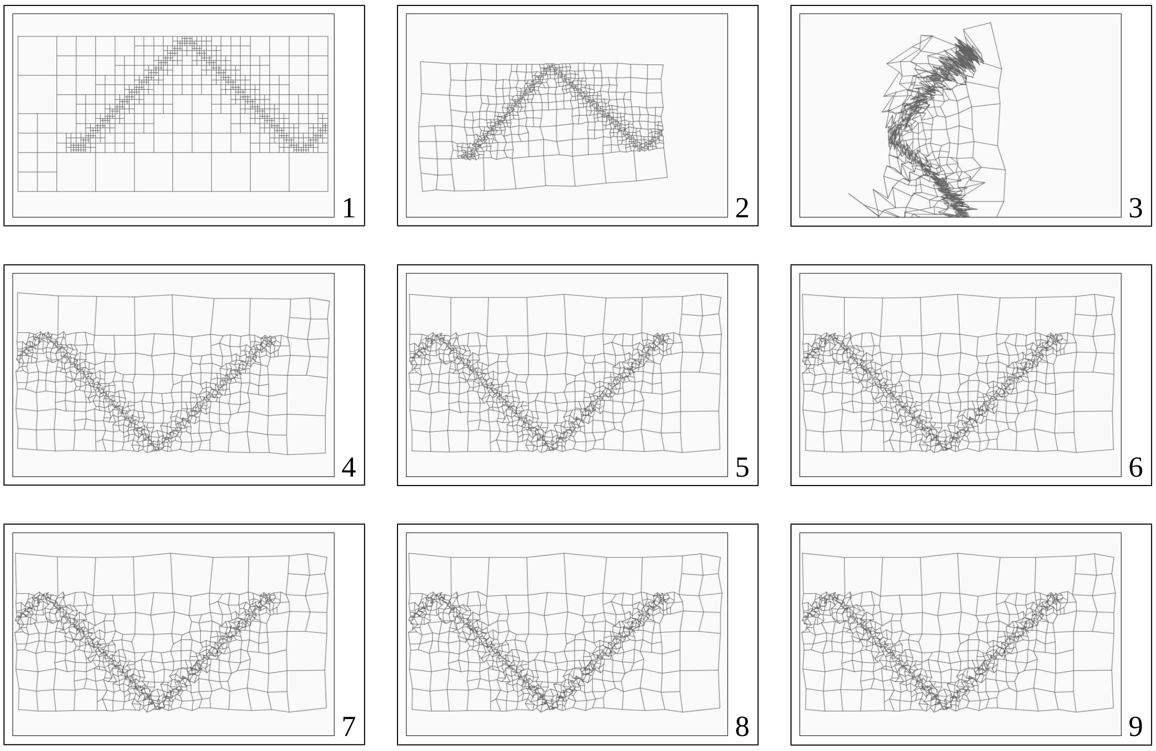
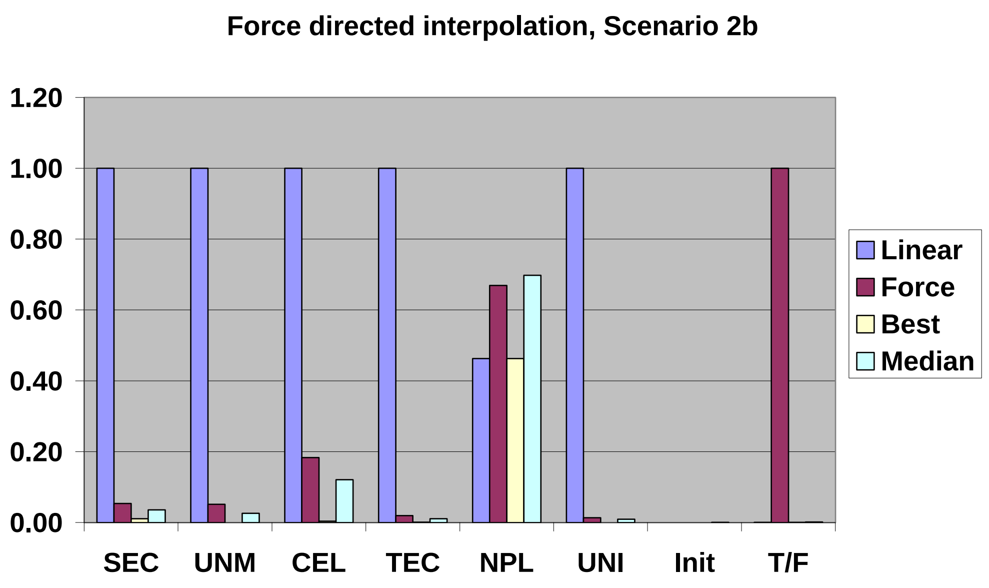


Figure 5.5: This figure shows 9 snapshots of a 101 frame animation. The animation is the result of applying the *Force Directed* method to scenario 2b.

results of the animation in regard to our measures are summarized in the following table:

Method	SEC	UNM	CEL	TEC	NPL	UNI	Init	T/F
Force	518	8.581E+08	37,058	11,203	635,514	47,201	0	3,647.9
Linear	9,681	1.670E+10	202,046	582,018	439,723	3,577,834	0	1.0
Best	105	6.805E+03	750	612	439,723	749	0	1
Median	347	4.339E+08	24,396	6,301	662,911	33,607	30	4.3



The results are very similar to the results for scenario 2. For the following scenarios we present the actual animation, the numeric values of the quality measures, as well as a graphical representation of the measures. However, we only comment on the results in cases where the *Force Directed* method shows behavior which is inconsistent with previous comments, or otherwise especially remarkable.

Scenario 3

Figure 5.6 shows the result of applying the *Force Directed* method to scenario 3. The numerical

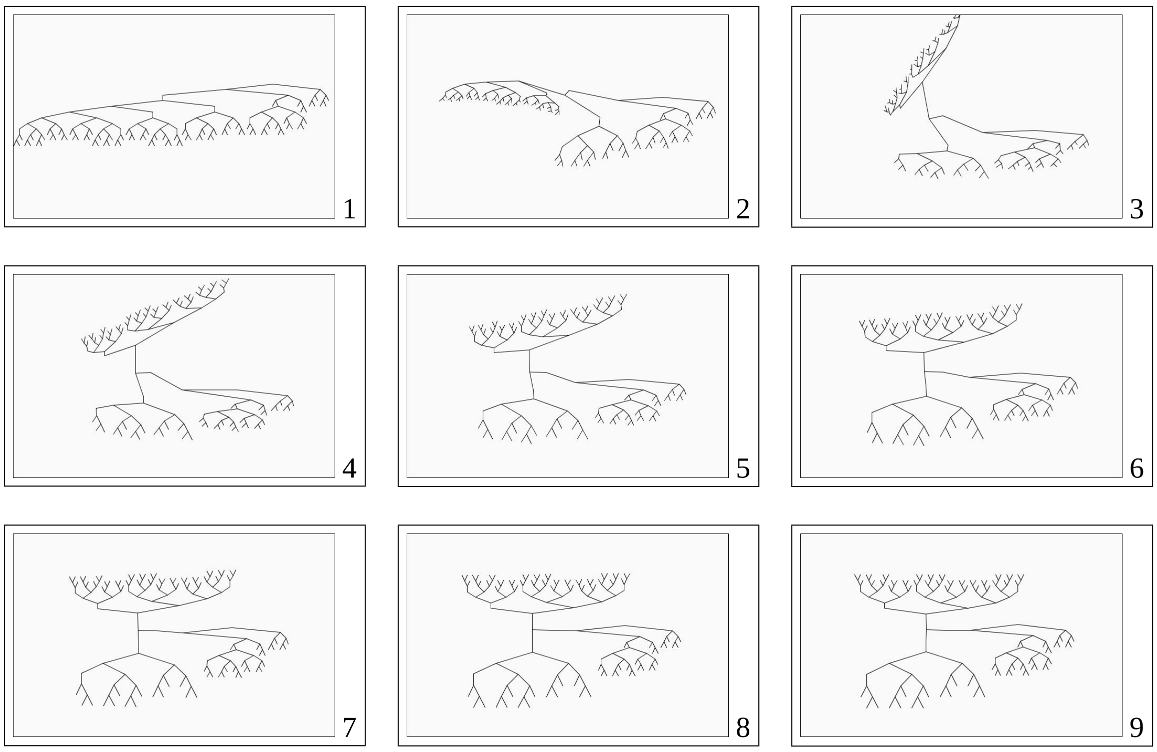
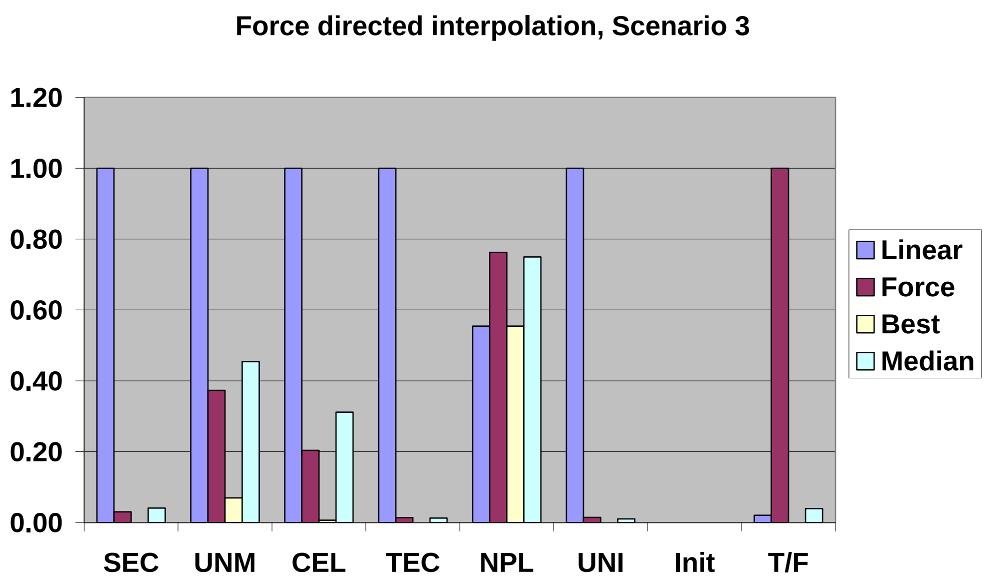


Figure 5.6: This figure shows 9 snapshots of a 101 frame animation. The animation is the result of applying the *Force Directed* method to scenario 3.

results of the animation in regard to our measures are summarized in the following table:

Method	SEC	UNM	CEL	TEC	NPL	UNI	Init	T/F
Force	1	1.477E+08	23,421	56	130,323	335	0	28.9
Linear	40	3.962E+08	114,892	4,055	94,772	23,519	0	0.6
Best	0	2.753E+07	720	0	94,772	0	0	0
Median	2	1.799E+08	35,772	51	128,102	245	0	1.1



The results for scenario 3 are consistent with the remarks for the previous scenarios.

Scenario 4

Figure 5.7 shows the result of applying the *Force Directed* method to scenario b4. The numerical

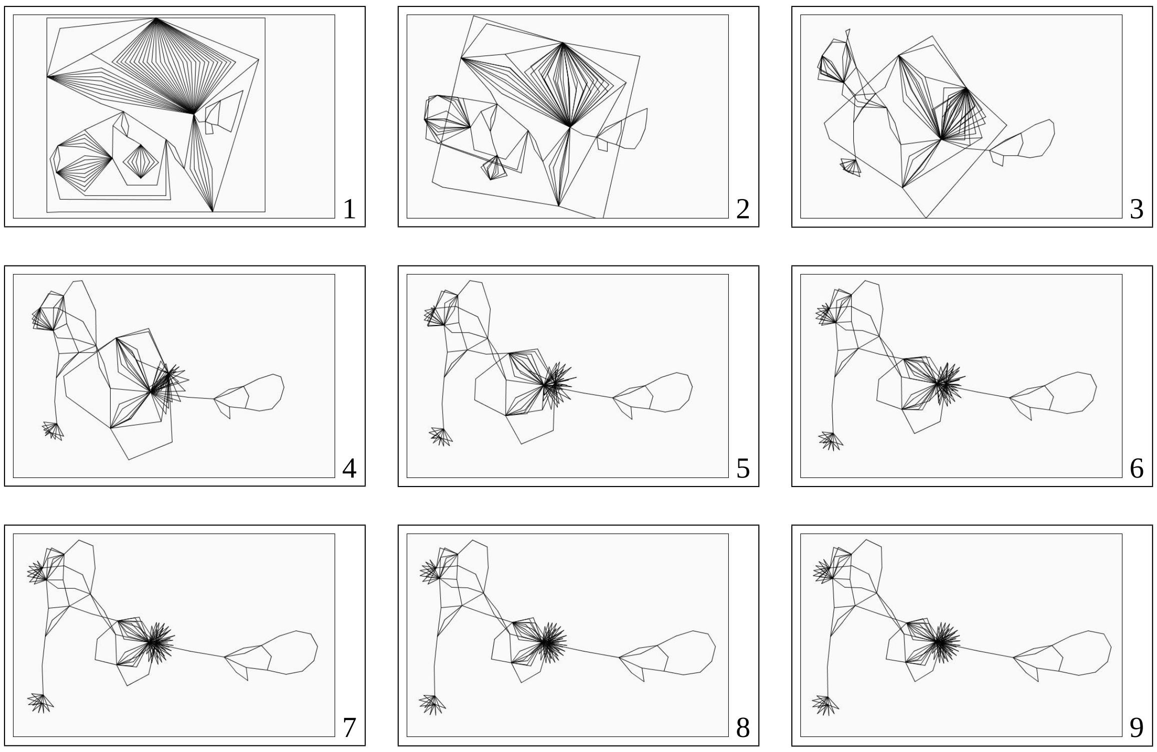
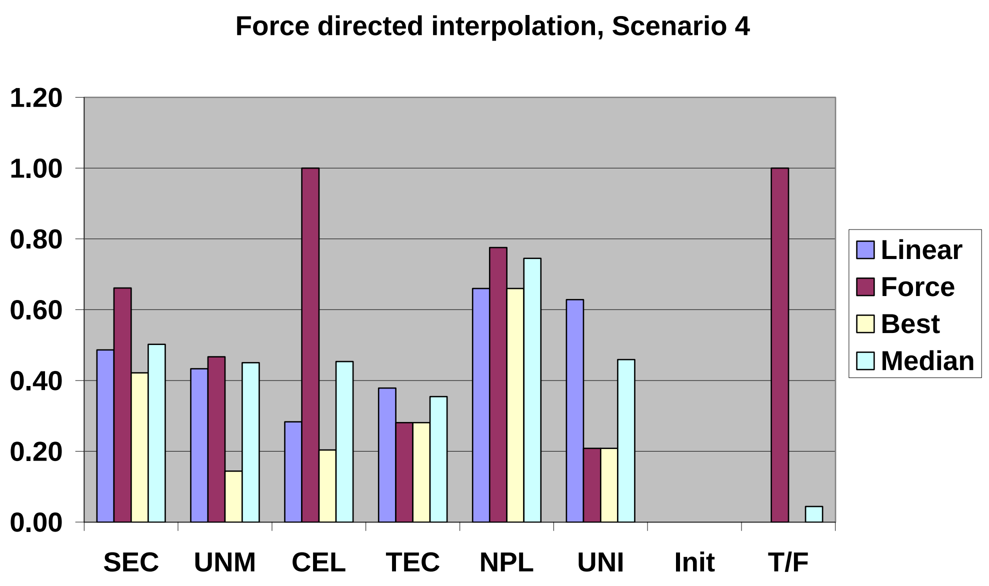


Figure 5.7: This figure shows 9 snapshots of a 101 frame animation. The animation is the result of applying the *Force Directed* method to scenario 4.

results of the animation in regard to our measures are summarized in the following table:

Method	SEC	UNM	CEL	TEC	NPL	UNI	Init	T/F
Force	258	3.104E+08	4,685,660	985	141,709	56	0	11.3
Linear	190	2.877E+08	1,326,461	1,328	120,561	169	0	0.0
Best	164	9.562E+07	954,237	985	120,561	56	0	0
Median	196	2.991E+08	2,125,756	1,244	136,133	124	0	0.5



The results for scenario 4 are consistent with the remarks for the previous scenarios.

Scenario 5

Figure 5.8 shows the result of applying the *Force Directed* method to scenario 5. The numerical

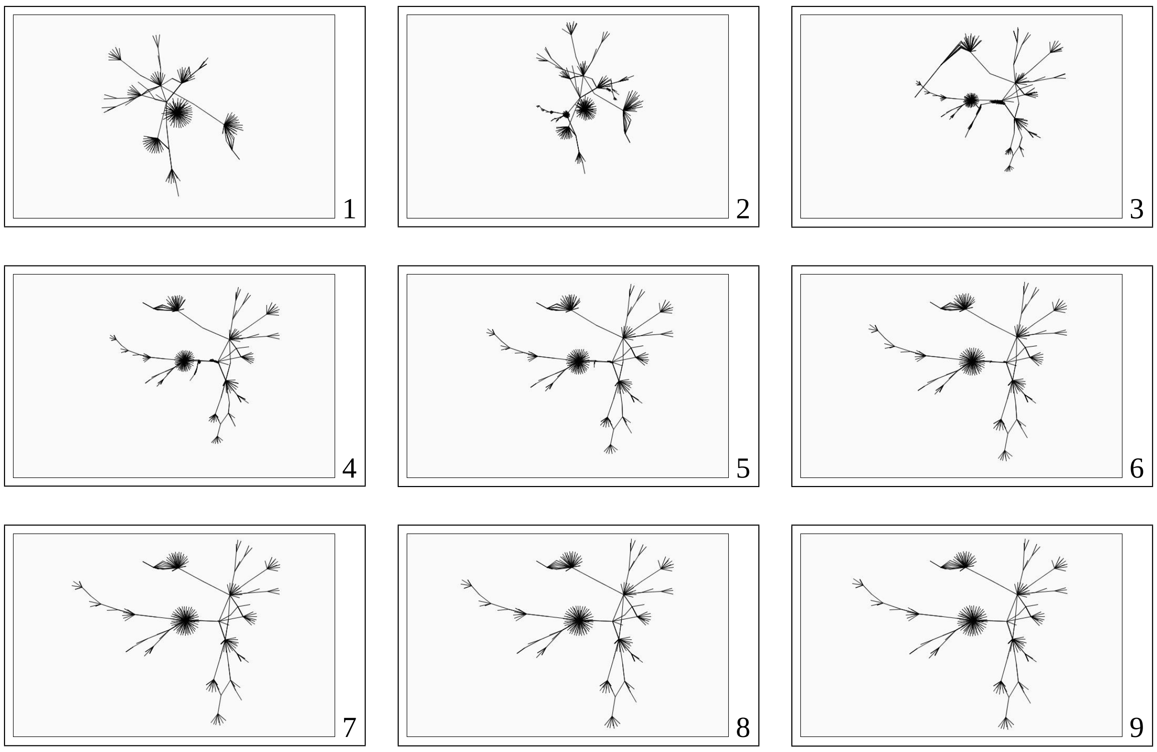
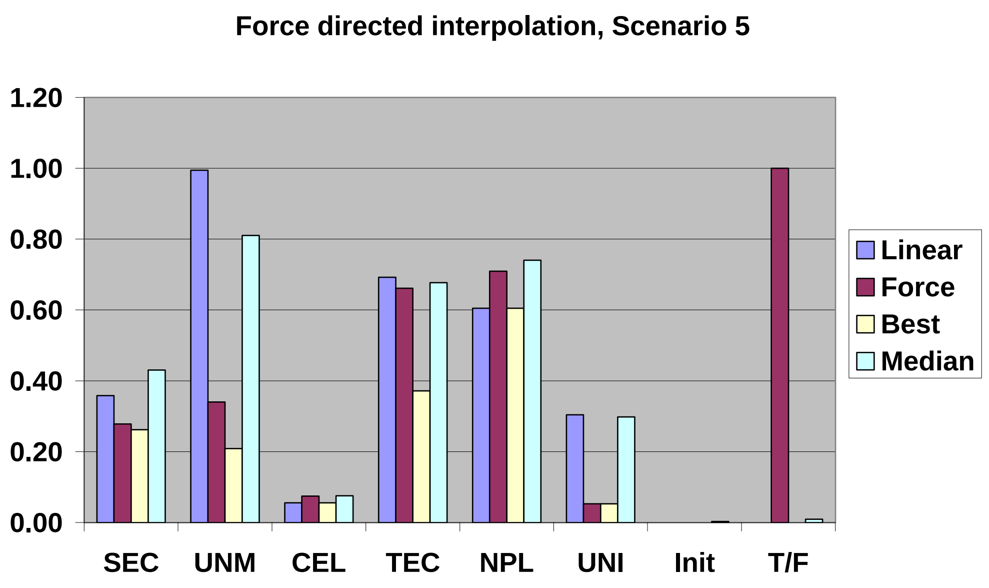


Figure 5.8: This figure shows 9 snapshots of a 101 frame animation. The animation is the result of applying the *Force Directed* method to scenario 5.

results of the animation in regard to our measures are summarized in the following table:

Method	SEC	UNM	CEL	TEC	NPL	UNI	Init	T/F
Force	327	5.029E+09	3,214,709	15,021	700,593	48	0	115.6
Linear	420	1.470E+10	2,399,401	15,725	597,271	277	0	0.0
Best	308	3.088E+09	2,399,401	8,438	597,271	48	0	0
Median	505	1.198E+10	3,262,048	15,373	731,150	272	55	1.1



The results for scenario 5 are consistent with the remarks for the previous scenarios.

Scenario 6

Figure 5.9 shows the result of applying the *Force Directed* method to scenario 6. The numerical

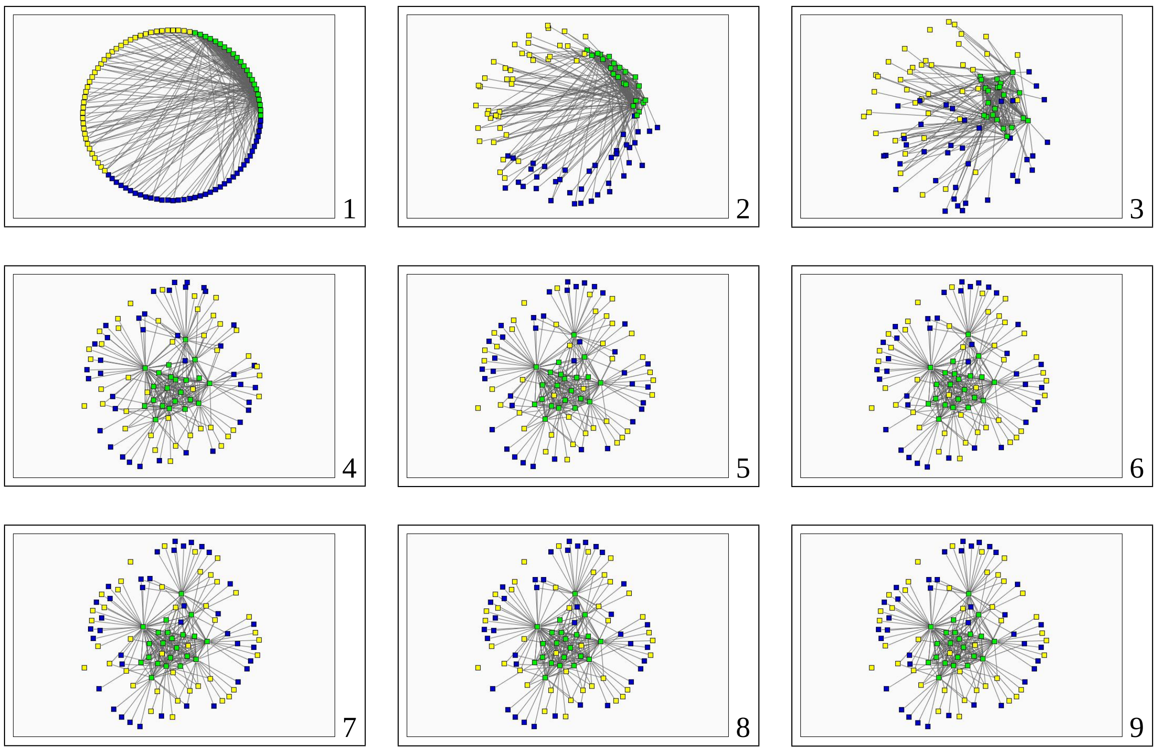
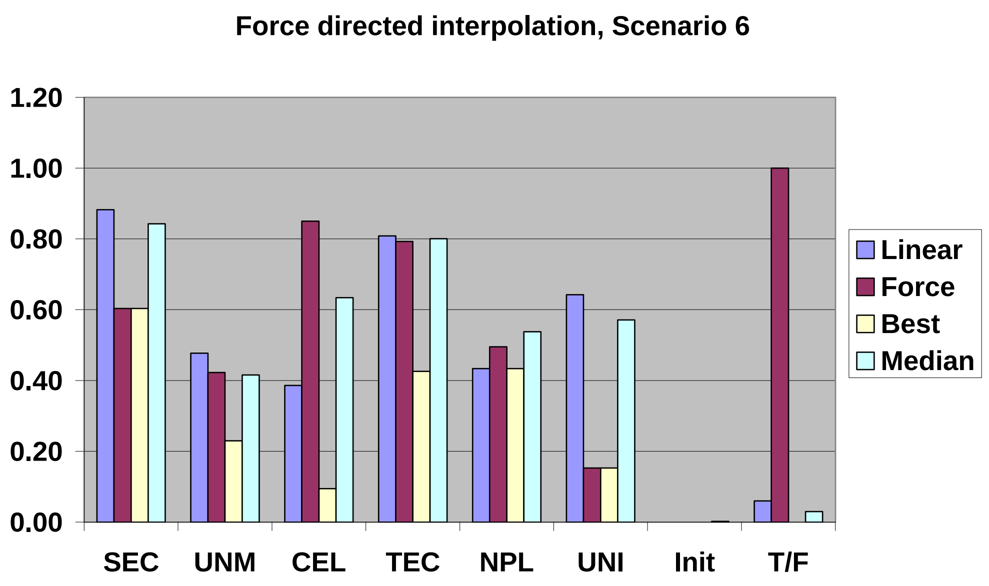


Figure 5.9: This figure shows 9 snapshots of a 101 frame animation. The animation is the result of applying the *Force Directed* method to scenario 6.

results of the animation in regard to our measures are summarized in the following table:

Method	SEC	UNM	CEL	TEC	NPL	UNI	Init	T/F
Force	4,853	2.111E+08	9,190,151	15,458	84,447	44	0	8.3
Linear	7,095	2.385E+08	4,174,533	15,762	73,957	185	0	0.5
Best	4,853	1.147E+08	1,020,039	8,301	73,957	44	0	0
Median	6,777	2.077E+08	6,856,478	15,610	91,672	165	25	0.2



The results for scenario 6 are consistent with the remarks for the previous scenarios.

Scenario 7

Figure 5.10 shows the result of applying the *Force Directed* method to scenario 7. The numerical

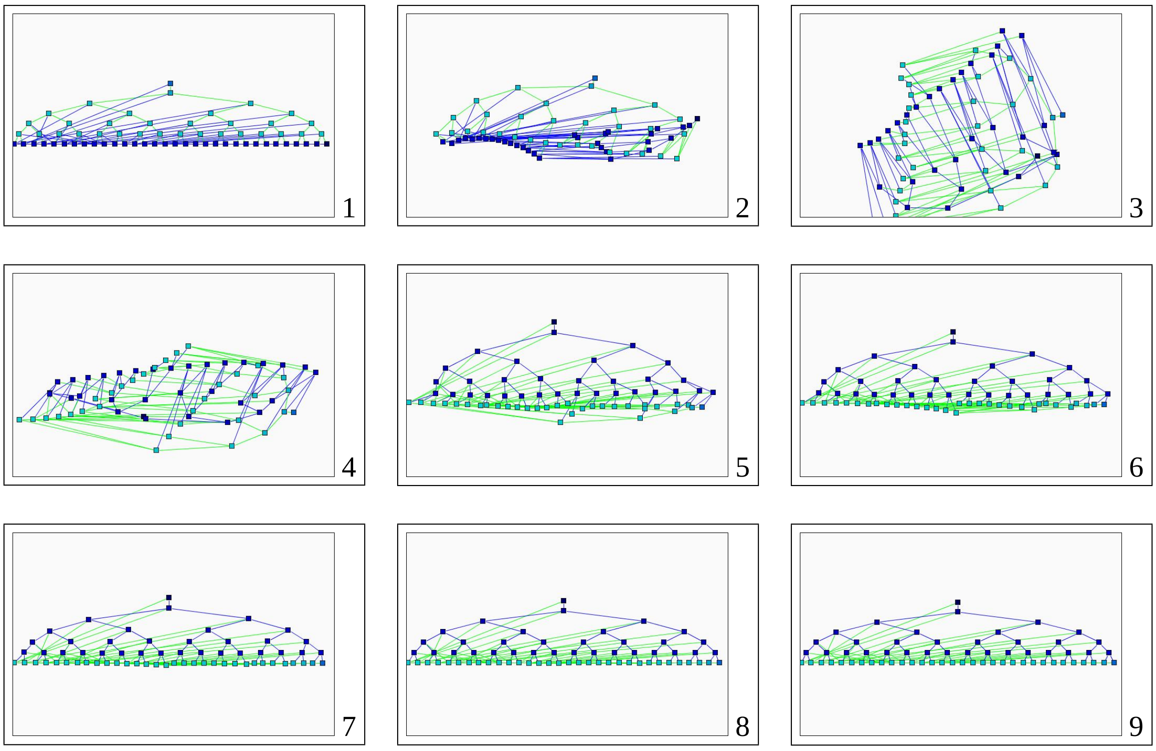
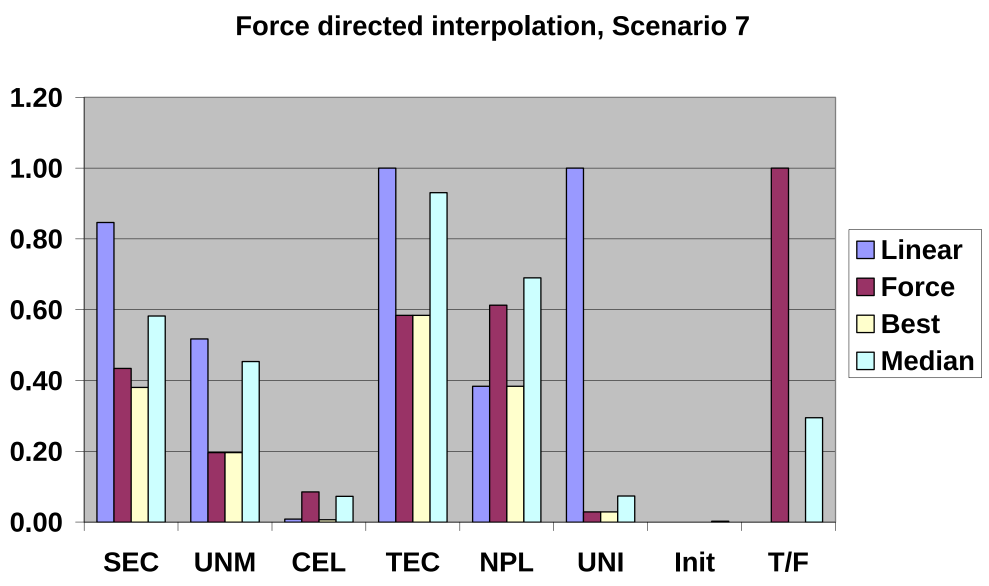


Figure 5.10: This figure shows 9 snapshots of a 101 frame animation. The animation is the result of applying the *Force Directed* method to scenario 7.

results of the animation in regard to our measures are summarized in the following table:

Method	SEC	UNM	CEL	TEC	NPL	UNI	Init	T/F
Force	518	1.841E+07	956,999	2,905	40,229	77	0	3.9
Linear	1,011	4.858E+07	90,948	4,975	25,170	2,692	0	0.0
Best	454	1.841E+07	71,857	2,905	25,170	77	0	0
Median	696	4.262E+07	816,052	4,631	45,283	198	25	1.1



The results for scenario 7 are consistent with the remarks for the previous scenarios.

Scenario 8

Figure 5.11 shows the result of applying the *Force Directed* method to scenario 8. The numerical

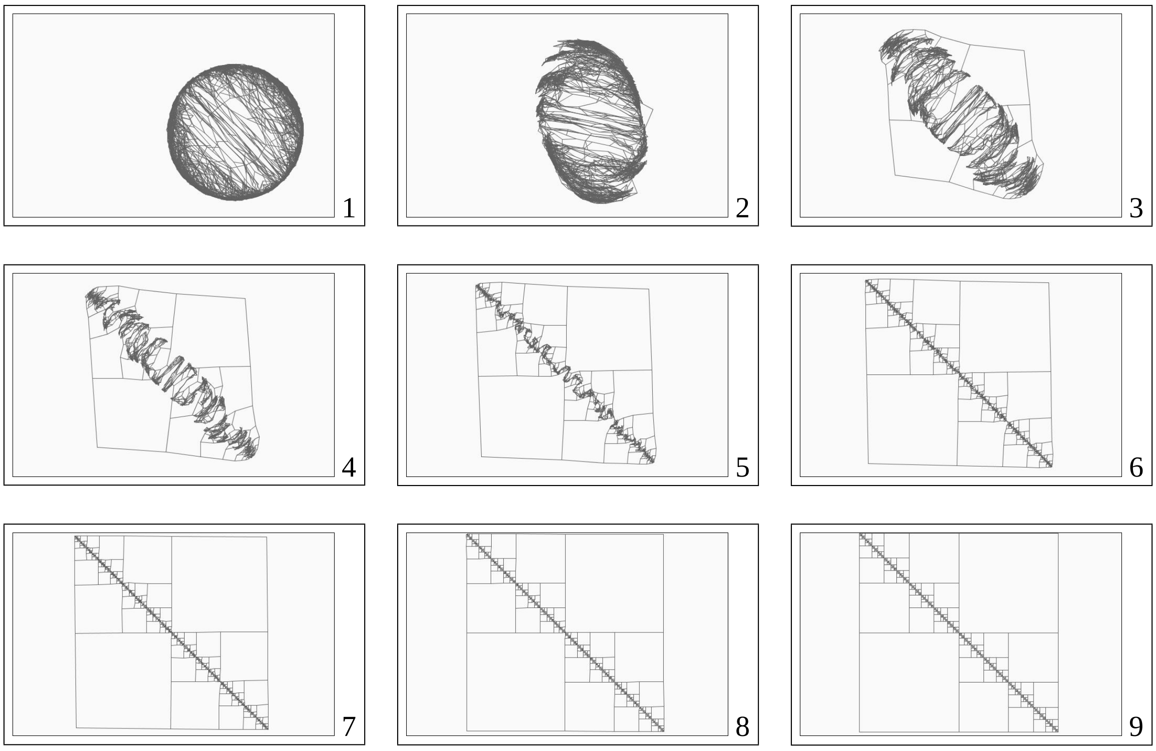
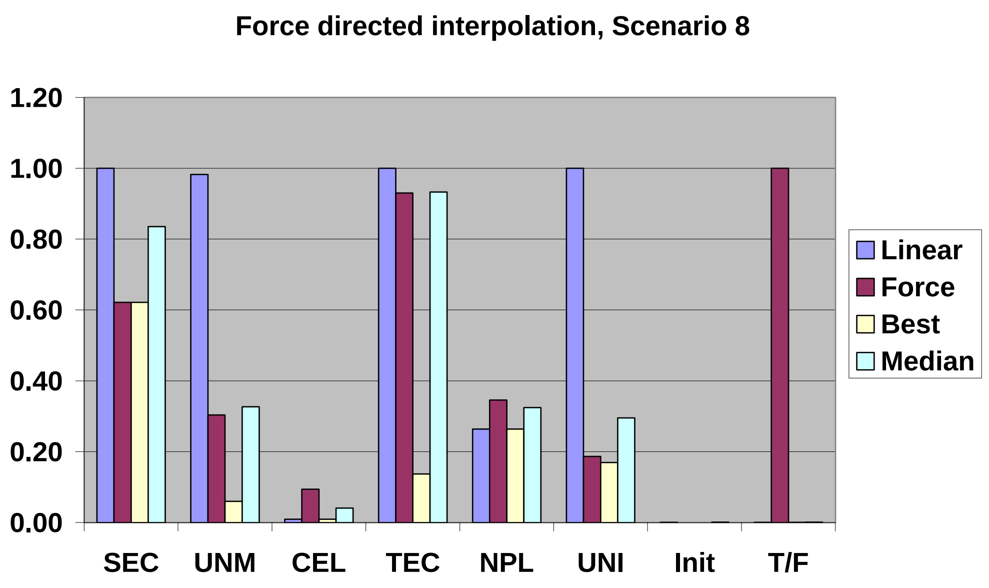


Figure 5.11: This figure shows 9 snapshots of a 101 frame animation. The animation is the result of applying the *Force Directed* method to scenario 8.

results of the animation in regard to our measures are summarized in the following table:

Method	SEC	UNM	CEL	TEC	NPL	UNI	Init	T/F
Force	28,706	8.743E+08	42,538	1,964,892	380,518	1,385,040	0	13,713.3
Linear	46,185	2.829E+09	4,138	2,112,739	290,510	7,427,591	50	7.2
Best	28,706	1.718E+08	4,138	289,169	290,510	1,256,611	0	4
Median	38,574	9.407E+08	18,486	1,971,003	357,359	2,192,830	80	13.7



The results for scenario 8 are consistent with the remarks for the previous scenarios.

5.1.4 Summary

The *Force Directed* method produces perceptually pleasing animations in almost all cases. It does not quite reach the same quality as specialized methods for certain specially structured scenarios, but still tends to produce acceptable animations in these cases.

The aesthetic quality of the *Force Directed* method is also reflected by good values over a wide range of measures and scenarios. We noticed, however, that the *Force Directed* method achieves comparatively poor values for *constant edge length* in some scenarios. This seems counter intuitive at first, as the *Force Directed* method tries to optimize *uniform node movement* and *constant edge length* is equivalent to *uniform node movement* when only applied to a subset of all node pairs, that is the edge set of the graph. The *Force Directed* method therefore, implicitly, also tries to optimize *constant edge length*. For most graphs the number of edges is linear to the number of nodes in the graph. Thus, in cases where there is an essential tradeoff between *constant edge length* and *uniform node movement*, we can expect the forces which support *constant edge length* to be opposed by a quadratic number of forces which support *uniform node movement*. This results in bad values for *constant edge length* in these cases.

Despite its high aesthetic qualities, the *Force Directed* method has several serious drawbacks. The most severe drawback is the high computational cost. In many of our benchmark tests[2], for example, the actual computational cost has been too high to allow the necessary frame rate of approximately 25 frames per second. As the *Force Directed* method takes $O\left(n^2\right)$ time to compute each frame, this problem cannot be solved by waiting for faster computers. The application of the *Force Directed* method is therefore limited to small graphs or non-interactive systems.

Further, the actual motion paths are only implicitly given by a set of non-linear equations. This set is not analytically solvable. As a consequence, the *Force Directed* method cannot provide random access to an arbitrary frame, it does not allow reversal of the direction of the animation[3], and it is impossible to effectively control the speed of the animation.

Many variations of the *Force directed* method are imaginable. It is possible to increase the emphasis on certain quality criteria by changing the weight ratio of the forces in the simulated system. It is also possible to introduce additional forces or modify current forces for this purpose. For example, the target attraction forces are currently modelled according to a spring force. Gravity forces could be used here instead. Care has to be taken in all cases, however, that it is guaranteed that all nodes end up at their target position at the end of the animation.

[2]Compare appendix A

[3]Unless all previous positions are explicitly recorded.

5.2 3D Spring Embedder Method

5.2.1 The Method

The major drawback of the previous approach (section 5.1) is its high time complexity. Each frame of the animation takes $O\left(n^2\right)$ computation time as each node has to be compared to each other node. This makes the method unsuitable for realtime systems. The computationally expensive node forces provide minimal distances between nodes during the animation, that is they act as a stabilizer of the proximity structure of the graph drawing. If we can achieve a similar effect by adding a small amount of edges to the graph we could avoid the time consuming computation of the node forces.

Furthermore, if we can compute the complete animation in the pre-processing phase of the animation, we can allow for a slightly worse than linear time complexity for the computation. To do this we build a three dimensional framework describing the animation. The $x - y$ plane of this three dimensional space is identical to the two dimensional geometric plane we use for drawing the graphs. The z axis represents the time. This means, we can see the initial drawing of the graph animation in the plane $(x, y, 0)$ and the target drawing of the graph animation in plane $(x, y, 1)$. The intermediate frames are located in the planes (x, y, z), $0 < z < 1$. We propose the following approach:

1. Add a small number of “stabilizing” edges to the graph.
2. Define a constant number of additional key frames.
3. Duplicate the whole graph for each additional key frame. Place each graph copy on a specific z coordinate.
4. Connect equivalent nodes between consecutive z-layers.
5. Set initial x and y coordinates for the nodes of each graph copy according to an interpolation method. The nodes in the first layer must lie on the start positions of the animation and the nodes of the last layer must lie on the target positions of the animation.
6. Simulate the edge forces in the three dimensional key frame graph. The nodes in the first and last frame are static. Nodes are not allowed to leave their z-plane.
7. The animation is shown as an interpolation along the key frames using a standard interpolation method.

Figure 5.12 shows an example. The initial and the target drawing are displayed in figure 5.12.1 and figure 5.12.2. Figure 5.12.3, 5.12.4, and 5.12.5 show snapshots of different views on the key frame graph. Different values of grey indicate different key frames. The key frames are arranged along the Z axis with all nodes of a key frame assigned to the same specific Z coordinate. No supporting edges were added in this example. The implementation of this method requires some issues to be

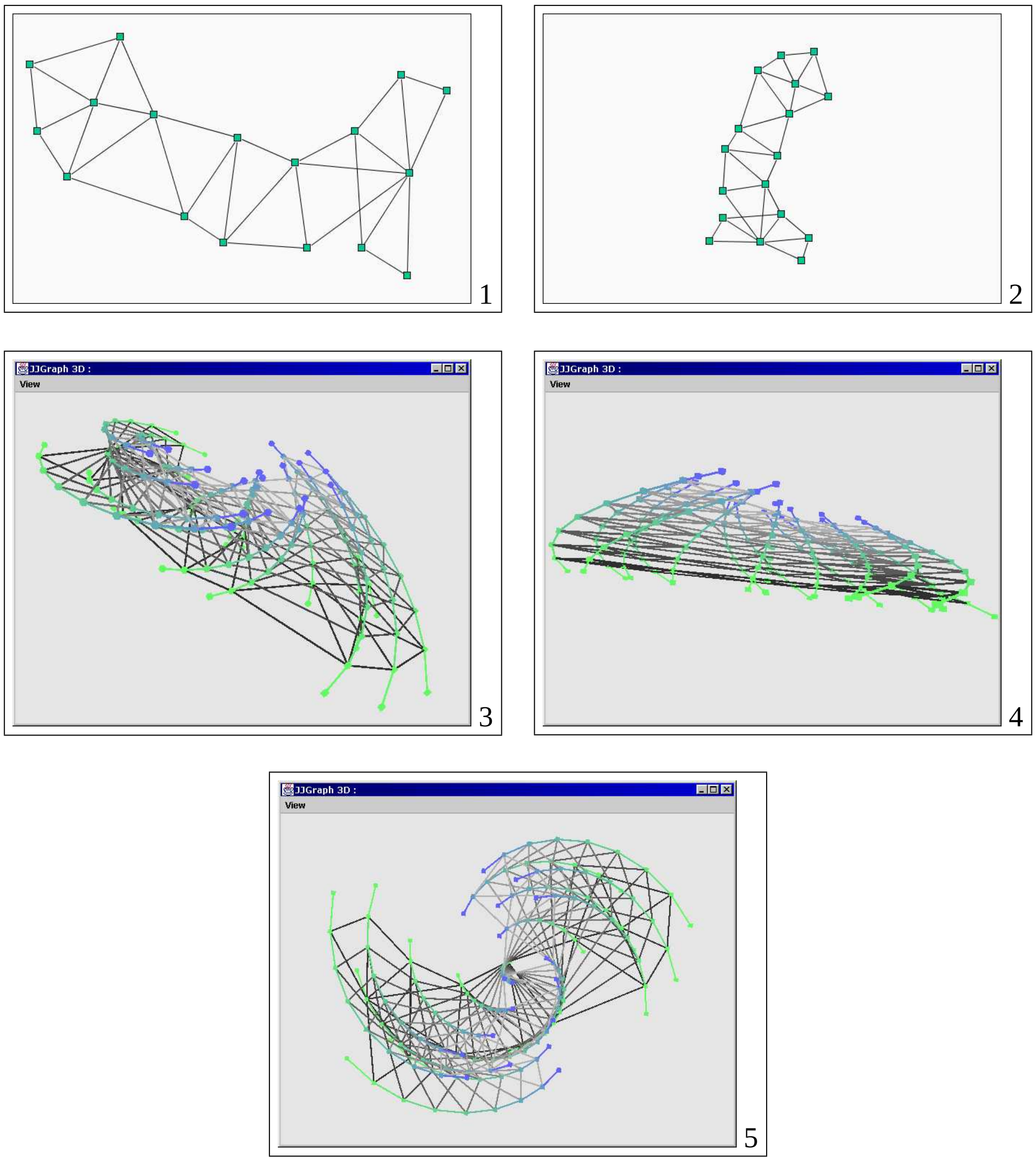


Figure 5.12: Force directed key frame generation

resolved at the following steps:

1. Add stabilizing edges to the graph.
2. Determine the number of key frames.
3. Compute initial positions for nodes in key graphs and assign optimal edge length to edges.
4. Simulate the system.
5. Interpolate key frames.

These are discussed below:

Adding stabilizing edges

The goal of adding stabilizing edges to the graph is to support structured movement of the graph during the animation. Experience has shown us that a “robust” graph structure is essential for creating structured motions when using the *3D Spring Embedder* method. We use the term “robust” to refer to the intuitive notion of robustness, that is the degree of resistance to deformation by outside forces. As the original graph structure might be arbitrarily “non-robust”, and in the worst case might not contain any edges at all, we have to augment the graph with supporting edges. In order to guarantee high computation speed the decision of where to add edges has to be made quickly and only a limited number of edges can be added to the graph. We used the following two heuristics:

1. Artificial center node

   An artificial node, called the *center node*, and edges from all other nodes to this node are added to the graph. The center node is placed centrally, for example at the barycenter or the center of the bounding box. We choose the barycenter for our implementation. The positions of the center nodes in each layer is on the interpolated direct line of the centers of the initial and target drawings. Figure 5.13 shows snapshots of three dimensional views of the resulting key frame graph. This procedure adds kn edges to the graph with k the number of key frames and n the number of nodes in the graph. The centers and their positions can be computed in $O(n)$. This step, therefore, does not add to the overall time complexity of the algorithm.

2. Add Delaunay Edges.

   A classical data structure from computational geometry, the *Delaunay triangulation*, connects points in the plane in a very compact, rigid form. The Delaunay triangulation $\mathcal{D}(S)$ of S is a

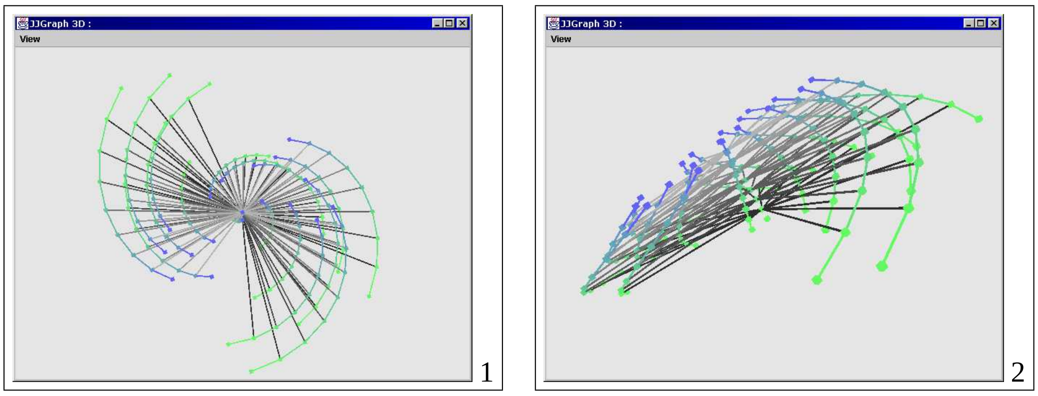


Figure 5.13: Frame graph with center node edges

planar graph drawing defined as follows: the nodes of $\mathcal{D}(S)$ consist of the data points of S, and two nodes s_i, s_j are joined by an edge if and only if there exists a circle passing through s_i, s_j having empty interior. The Delaunay triangulation is the graph dual of the well-known Voronoi diagram of S. Some of the interesting features of Delaunay triangulations are listed below; more details can be found in virtually any textbook on computational geometry, for example [26].

(a) If s_i is the nearest neighbor of s_j from among the data points of S, then (s_i, s_j) is an edge in $\mathcal{D}(S)$.

(b) The number of edges in $\mathcal{D}(S)$ is at most $3n - 6$, and thus the average number of neighbors of a site s_i in $\mathcal{D}(S)$ is less than 6.

(c) The Delaunay triangulation is the most well-proportioned over all triangulations of S, in that the size of the minimum angle over all its triangles is the maximum possible.

(d) The triangulation $\mathcal{D}(S)$ can be robustly computed in $O(n \log n)$ time.

Figure 5.14 shows the Delaunay triangulation of the nodes of the graph from figure 5.12.1. The Delaunay triangulation adds $O(kn)$ edges to the graph with k being the number of key frames and n being the number of nodes in the graph. The Delaunay triangulation has to be computed only once, taking $O(n \log n)$ time. This increases the preprocessing time to $O(n \log n)$ and leaves the frame generation time at $O(n)$. Figure 5.15 shows two snapshots of the three dimensional key frame graph with Delaunay edges. The Delaunay edges increase the “robustness” of the graph structure significantly. There is a path from the interior of each Delaunay triangle to the interior of each other Delaunay triangle through adjacent Delaunay

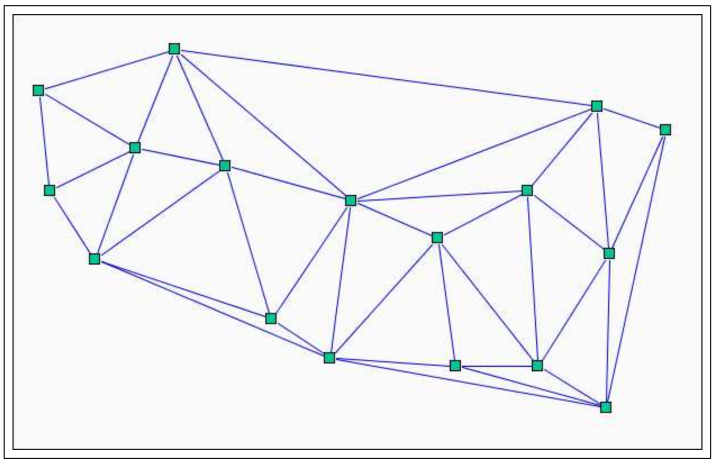


Figure 5.14: Example of the Delaunay Triangulation of the graph from figure 5.12.1.

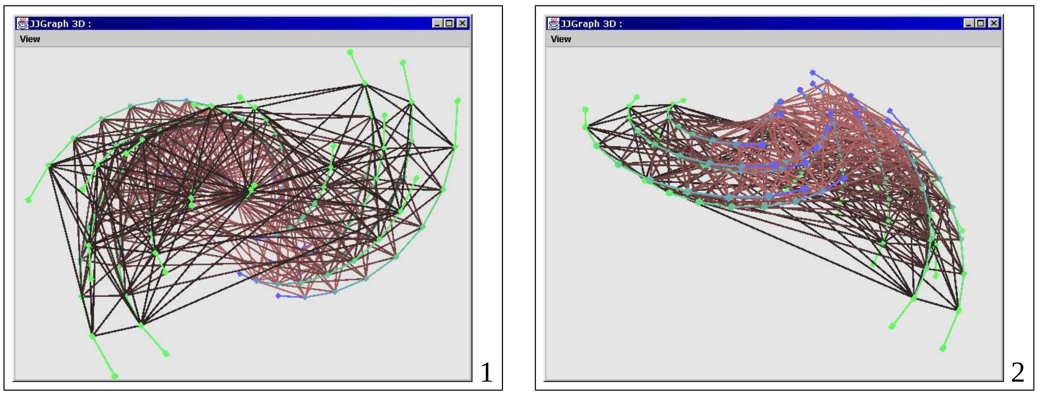


Figure 5.15: Frame graph with Delaunay edges

triangles. The Delaunay framework is thus rigid. This means that we cannot deform the graph in any way which changes angles or distances between nodes without increasing the length of a Delaunay edge[4]. It is well known from civil engineering that the "robustness" of a triangle is in direct relation to the minimum angle of the triangle. The Delaunay triangulation guarantees that the size of the minimum angle over all its triangles is the maximum possible. This means that in this sense it maximizes the "robustness" of the framework.

Determine number of key frames

Every key frame adds a linear number of nodes and edges to the key frame graph. Any small constant number of key frames is, therefore, acceptable. We use 10 key frames in our implementation.

[4]For a detailed discussion on the concept of "rigidity" see, for example, [23]

Compute initial positions for key graphs and edge lengths

The choice of good initial positions of the nodes of the key frame graph is essential for a fast convergence of the edge force simulation. Several approaches are possible. For example, the nodes of the key frame graphs could be placed on the direct line between the positions of the corresponding nodes in the initial and the target drawing. In our implementation of this method we apply a slightly more sophisticated approach, however. We compute a transformation for the whole graph using the linear regression method from chapter 6 and place the nodes according to this transformation. For the scenario in figure 5.12 the linear regression method would find a rotation by 90^o and a scaling by a factor of 0.5. Figure 5.16 shows the initial placement of the nodes according to this scenario. Each edge in the key frame graph has to be assigned an optimal length and a weight. Many different

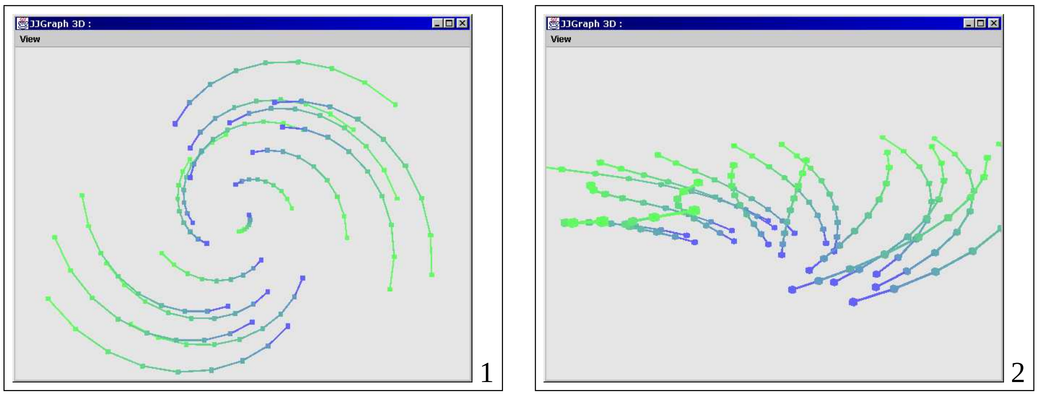


Figure 5.16: Frame graph with node positions set according to linear regression analysis.

approaches are possible. We use the fact that we have four different kinds of edges in our graph: the original graph edges, the Delaunay edges, the center node edges, and the edges between the key frames. For the original graph edges, the Delaunay edges, and the center node edges we use the linearly interpolated edge length between the length in the initial and in the target drawing as optimal edge length in each key graph. These forces encourage the nodes to move in a structured way, preserving relative distances. We use the distance between the key frame graphs on the z axis as the optimal length for the inter key frame graph edges. These forces try to keep the path of node movements as short as possible. We assign an individual weight to each class of edge and assign the same weight to each edge within the class. A good choice of weights is not easy and we found different weight distributions to be optimal in different scenarios. Developing heuristics to automatically determine good weights for a given scenario is as yet an unsolved problem. We use the weights shown in the following table for our experiments.

Edge type	Weight
Graph edges	1.0
Delaunay edges	1.0
Center node edges	1.0
Inter key frame edges	0.7

Simulation of the forces

We simulate the system by iteratively updating the node positions according to the edge forces. The simulation stops if the system enters a stable configuration or a maximum number of iterations has been performed. Given a key frame graph $G = (V, E_G \cup E_D \cup E_C \cup E_I)$ with a set E_G of original graph edges, a set E_D of Delaunay edges, a set E_C of center node edges, and a set E_I of edges between key frame graph layers, as well as a distance function $d : (V, V) \to \mathbb{R}$, a length function $l : E \to \mathbb{R}$ assigning an optimal length to each edge, and a weight function $w : \mathbb{P}(E) \to \mathbb{R}$ assigning a weight to each class of edges, the following force applies to each node $v \in V$:

$$Force\,(v) = \frac{\sum_{k\in\{G,D,C,I\}} w\,(E_k) \sum_{e=(u,v)\in E_k} \frac{u-v}{d(u,v)} \left(d\,(u,v) - l\,(e)\right)}{\sum_{k\in\{G,D,C,I\}} w\,(E_k)\,|E_k|}$$

Interpolate key frames

During the animation the nodes have to be moved along a path interpolating the key frame positions. Standard point interpolation techniques can be used. For our scenarios we move the nodes along the direct line between key frame positions. As an alternative NURBS could be used to provide a smooth interpolation of the node paths, as it is not essential that all key positions are exactly met and NURBS are invariant to simple transformations preserving structured movements of nodes.

5.2.2 Analysis

Straightforward modifications of various existing spring embedder algorithms [7] can be used to perform the simulation of the physical force system. Each of these algorithms has slightly different properties and the analysis of our measures thus theoretically depends on which method is used. To allow for a more general discussion we only make a few assumptions on the chosen method. We assume that the chosen method converges to a local minimum, that is to a configuration where the kinetic energy of the nodes and the sum of the forces for each node are zero. Convergence can usually easily be guaranteed by applying the appropriate physical laws and using a dampening factor. We further assume that the method is able to compute the edge forces in a time linear to the number of edges in the graph.

1. Init

   Given a graph $G = (U, V)$ and $n_v = |V|$, $n = |V| + |E|$, k key frames, and an upper bound i on the number of iteration of the simulation, the following steps have to be performed:

   - Determine the center of the graph ($O(n_v)$), add center nodes $O(1)$, add edges from all nodes to the center node $O(n_v)$.
   - Compute the Delaunay triangulation ($O(n_v \log n_v)$) and add Delaunay edges to the graph ($O(n_v)$). This take a total time of $O(n_v \log n_v)$.
   - Copy the graph for each key frame ($O(kn)$), add edges between key frames ($O(kn)$).
   - None of the previous steps adds more than $O(n_v)$ edges to the graph. During each iteration of the simulation we have to update all node positions according to the edge forces. As each edge is connected to exactly two nodes this can be done in $O(n)$ time. The number of iterations of the simulation is bounded by a constant i, after which we terminate the simulation if no stable configuration has been reached. The complete simulation, therefore, takes $O(ikn)$ time.

   This amounts to a total *Init* time of $O(n_v \log n_v) + O(kn)$. With k and i constant this gives an *Init* time of $O(n \log n)$ which is an acceptable time complexity

2. T/F

   The position of every node can be updated in constant time, resulting in a time complexity of $O(n)$ to compute a frame.

3. UNI

   The *3D Spring Embedder* method does not consider inter-node distances explicitly. It does however provide some mechanisms which tend to reduce *unnecessary node intersections*. The graph-edge forces help to avoid intersections of nodes which are connected by an edge. The artificial Delaunay edges tend to reduce *unnecessary node intersections* of nodes which lie close to each other in the initial drawing. As a consequence, the *3D Spring Embedder* method generally achieves good values for *unnecessary node intersections* in scenarios which involve dense graphs, as well as in scenarios which preserve the topology of the layout between the initial drawing and the target drawing. For scenarios involving sparse graphs, however, once the topology is broken the stabilizing mechanisms cease to work and are thus no longer able to effectively avoid node intersections.

4. CEL

   The graph-edge forces encode this measure explicitly in the simulation. Further, all other edges in the physical system are also directly related to one or more measures. The computed animation corresponds to a local energy minimum of the simulated force system. We therefore know that there can be no other animation in the immediate vicinity of the computed animation which has better values for *constant edge length* and at least as good values for all other measures. By adjusting the weight of the graph edge forces more or less emphasis can be put on achieving good values for *constant edge length*.

5. Smoothness

   As the direction of node paths can change abruptly at each key frame position, the animation is generally not smooth. The inter plane edges encourage smooth transitions to some degree, but do not guarantee them. Spline path interpolation can be used to guarantee smooth transitions over the key frames. Care has to be taken, however, not to increase the value of other measures by doing so, as many spline interpolation techniques are not invariant to simple transformations such as rotations.

6. UNM

   The Delaunay edges and the original graph edges support structured movements. For most graphs in real applications the number of original edges is linear to the number of nodes in the graph. The Delaunay triangulation only adds a linear number of edges to the graph. In a typical scenario, thus, only the square root of the number of node pairs contributing to the *uniform node movement* measure are represented by forces in the simulated system; the majority of the node pairs are not represented at all. One might expect this to lead to poor values for *uniform node movement*. However, the support framework which is built by the Delaunay edges is exceptionally robust to deformations, as explained above. We will see in the empirical evaluation (Section 5.2.3) that, in fact, the Delaunay framework is able to supply sufficient stability to the graph structure to guarantee very good values for *uniform node movement* in all scenarios.

7. NPL

   The paths of the nodes are described by the inter-plane edges. Similar to *constant edge length*, as the computed animation corresponds to a local energy minimum of the simulated force system, we know that there is no animation in the vicinity of the computed animation which has better values for *node path length* and at least as good values for all other measures. By adjusting the weight of the inter-plane edge forces, more or less emphasis can be put on achieving good values for *node path length*.

5.2.3 Empirical Evaluation

Scenario 1

Figure 5.17 shows the result of applying the *3D Spring Embedder* method to scenario 1. The

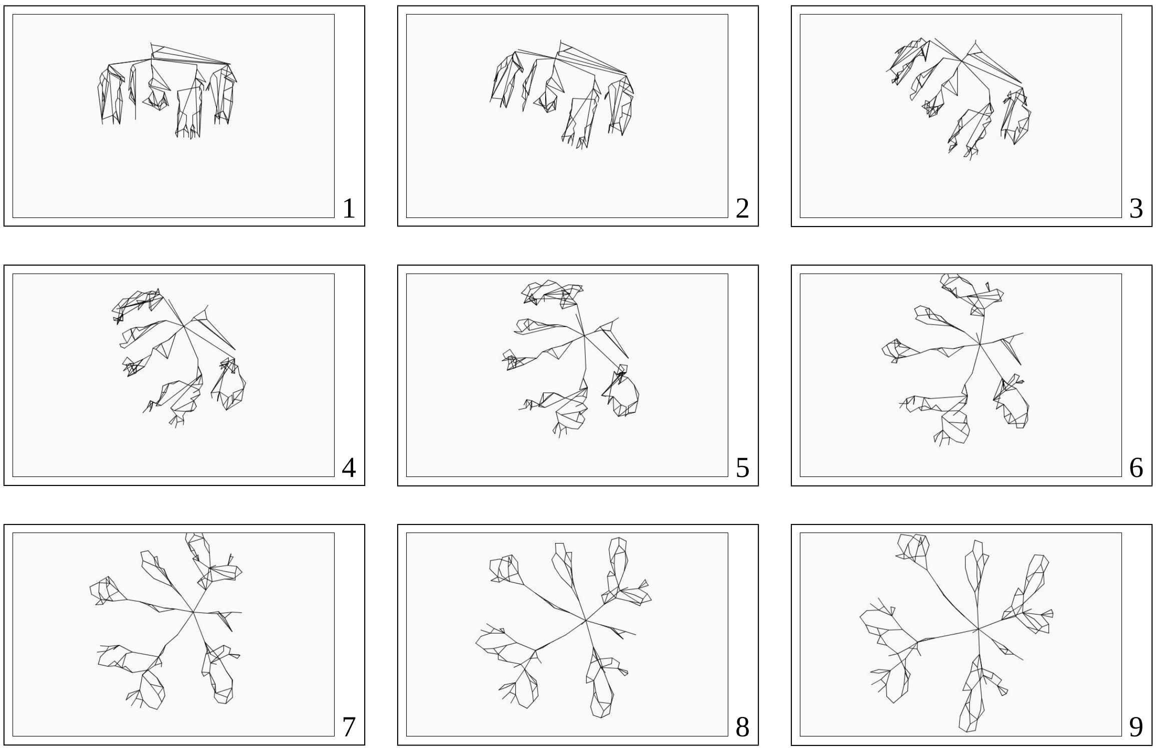
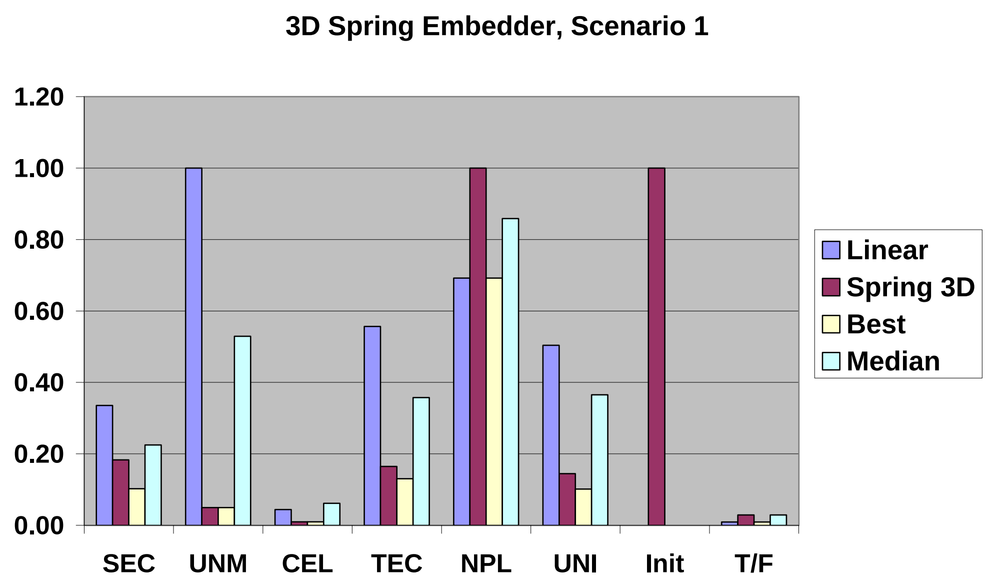


Figure 5.17: This figure shows 9 snapshots of a 101 frame animation. The animation is the result of applying the *3D Spring Embedder* method to scenario 1.

numerical results of the animation in regard to our measures are summarized in the following table:

Method	SEC	UNM	CEL	TEC	NPL	UNI	Init	T/F
Spring 3D	71	2.484E+08	402,170	621	502,950	37	32,350	1.6
Linear	130	5.020E+09	1,843,276	2,095	347,983	129	0	0.5
Best	40	2.484E+08	402,170	490	347,983	26	0	0
Median	87	2.656E+09	2,577,963	1,346	431,802	94	0	1.6



As expected, the time to initialize the animation is much higher than for any other method. Also, *node path length* is slightly higher than for the other methods. On all other measures however, the *3D Spring Embedder* method achieves very good values.

Scenario 1b

Figure 5.18 shows the result of applying the *3D Spring Embedder* method to scenario 1b. The

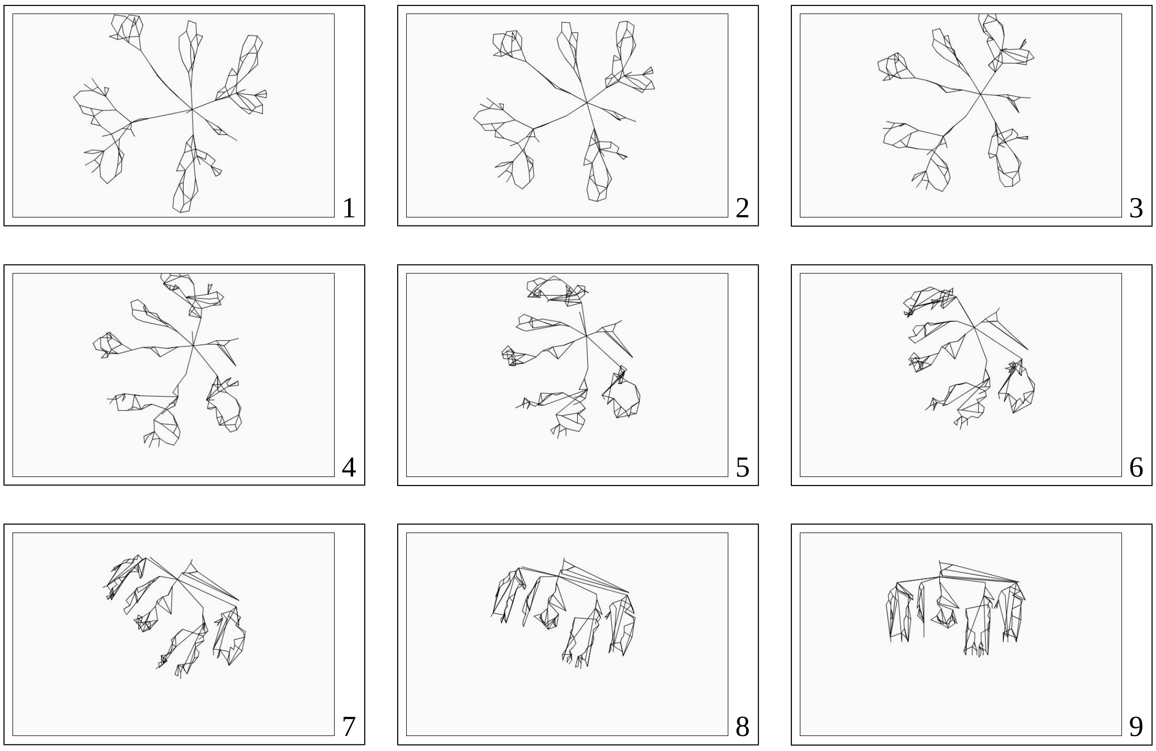
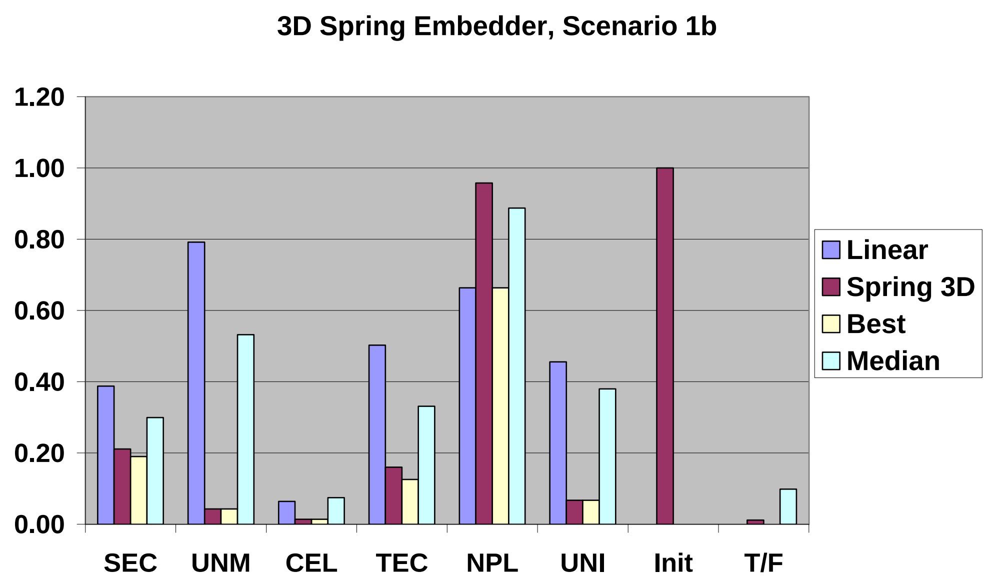


Figure 5.18: This figure shows 9 snapshots of a 101 frame animation. The animation is the result of applying the *3D Spring Embedder* method to scenario 1b.

numerical results of the animation in regard to our measures are summarized in the following table:

Method	SEC	UNM	CEL	TEC	NPL	UNI	Init	T/F
Spring 3D	71	2.697E+08	395,880	680	502,447	19	37,080	0.6
Linear	130	5.020E+09	1,843,276	2,138	347,983	129	0	0.0
Best	64	2.697E+08	395,880	533	347,983	19	0	0
Median	100	3.371E+09	2,146,880	1,409	465,534	108	0	5.1



The *3D Spring Embedder* method is theoretically symmetrical for exchanged initial and target drawings. The small deviations of the obtained measures compared to the results for scenario 1 can be explained by the inherent non-determinism of most force directed algorithms, as well as by numerical inaccuracies.

Scenario 2

Figure 5.19 shows the result of applying the *3D Spring Embedder* method to scenario 2. The

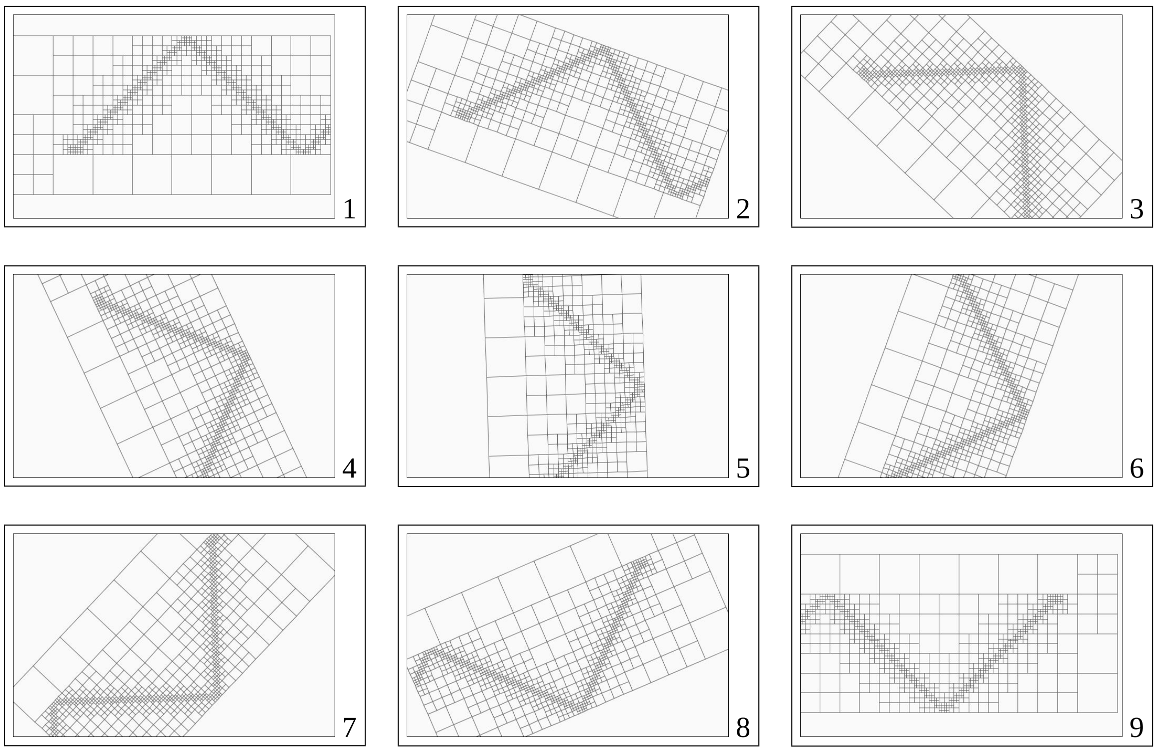
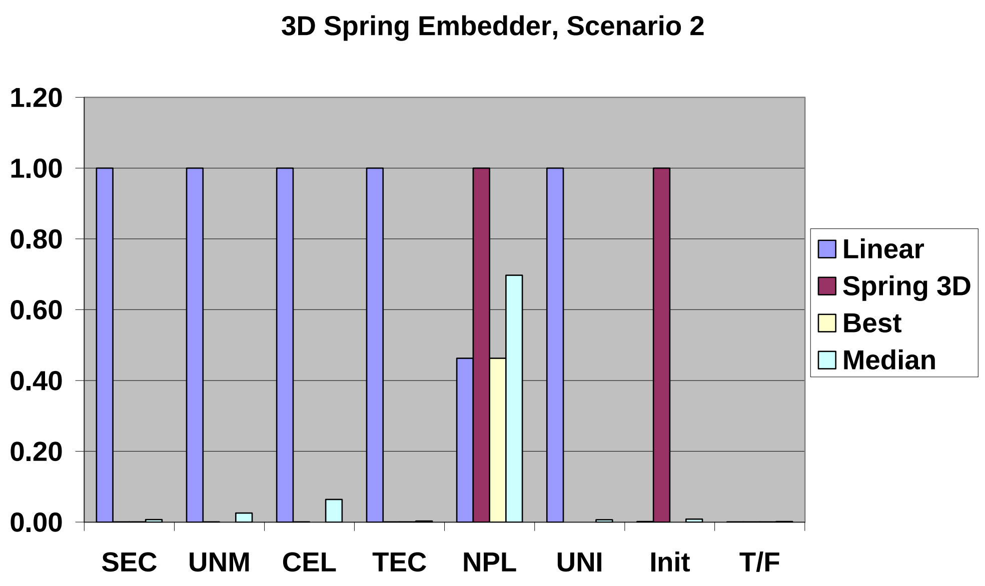


Figure 5.19: This figure shows 9 snapshots of a 101 frame animation. The animation is the result of applying the *3D Spring Embedder* method to scenario 2.

numerical results of the animation in regard to our measures are summarized in the following table:

Method	SEC	UNM	CEL	TEC	NPL	UNI	Init	T/F
Spring 3D	4	9.863E+06	125	374	950,648	0	32,690	1.1
Linear	12,891	1.675E+10	212,992	1,301,941	440,110	3,813,032	50	1.6
Best	4	1.066E-20	0	374	440,110	0	0	1
Median	92	4.259E+08	13,608	3,468	662,794	24,579	270	4.7



[5]

Apart from Init time and *node path length*, the *3D Spring Embedder* method achieves results which are several orders of magnitude better than the *Linear Interpolation* method. The results are close to the optimum in most cases. However, for *uniform node movement* the result is significantly worse than the best result. Although the snapshots of the animation (Figure 5.19 are almost identical to the snapshots of the optimal method (Figure 6.8, page 217), a look at the actual animation reveals the reason for the bad value. The node paths are determined by piecewise direct linear interpolations between the virtual key frames. This means, that although the *3D Spring Embedder* can generate the correct virtual key frames for the 180^o rotation, the paths between the virtual key frames are line segments and not arcs. This results in the relatively bad values for *uniform node movement* and induces the impression of a repeatedly expanding and shrinking graph during the animation.

[5]Note that the bar chart is misleading for *uniform node movement* in this case. In the chart it looks as if the value for *uniform node movement* is close to the optimum. This is actually not true.

Scenario 2b

Figure 5.20 shows the result of applying the *3D Spring Embedder* method to scenario 2b. The

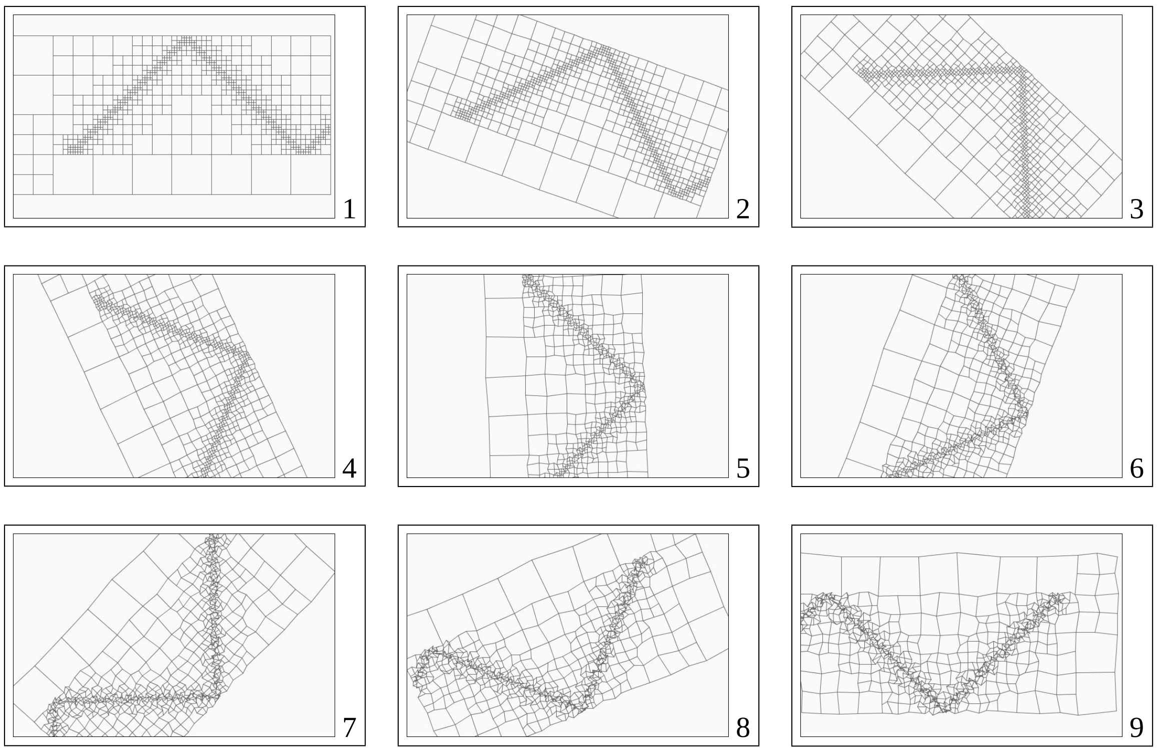
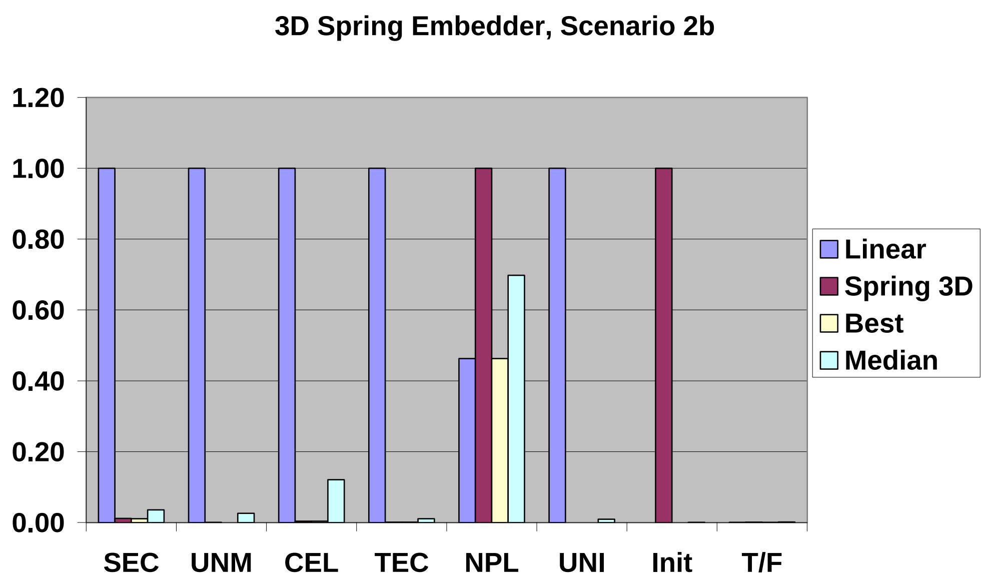


Figure 5.20: This figure shows 9 snapshots of a 101 frame animation. The animation is the result of applying the *3D Spring Embedder* method to scenario 2b.

numerical results of the animation in regard to our measures are summarized in the following table:

Method	SEC	UNM	CEL	TEC	NPL	UNI	Init	T/F
Spring 3D	114	9.687E+06	750	623	949,956	749	122,650	2.8
Linear	9,681	1.670E+10	202,046	582,018	439,723	3,577,834	0	1.0
Best	105	6.805E+03	750	612	439,723	749	0	1
Median	347	4.339E+08	24,396	6,301	662,911	33,607	30	4.3



The same remarks as for scenario 2 apply to scenario 2b.

Scenario 3

Figure 5.21 shows the result of applying the *3D Spring Embedder* method to scenario 3. The

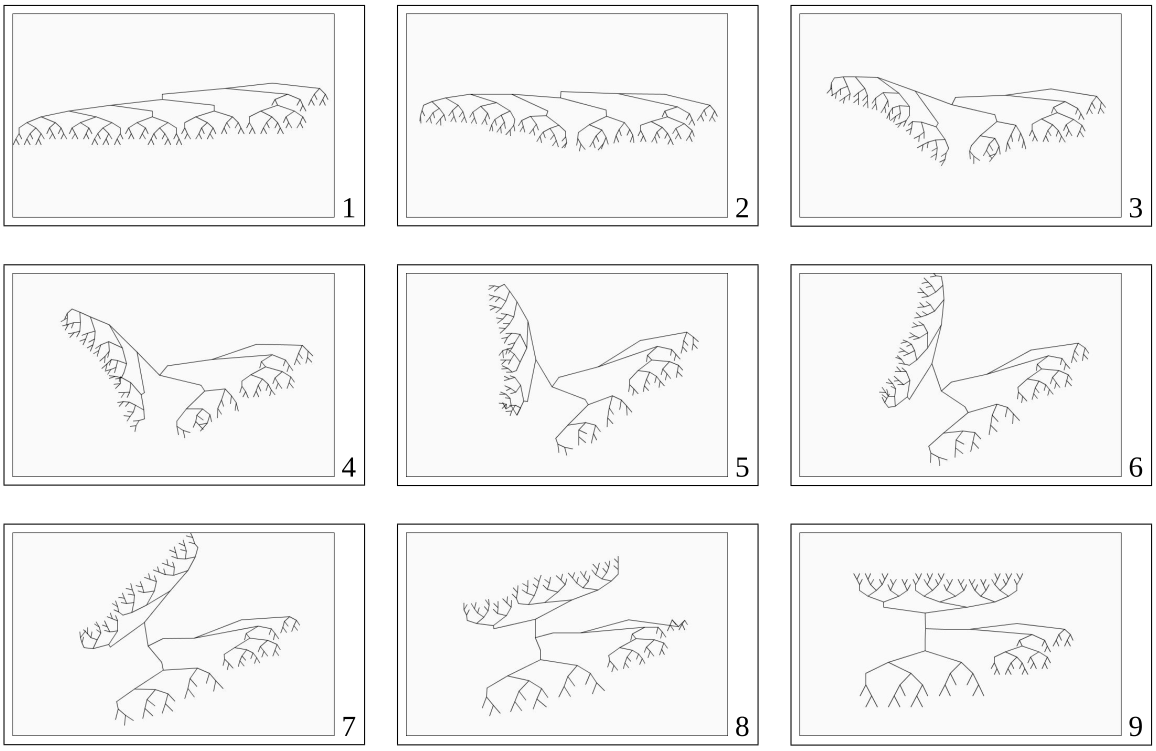
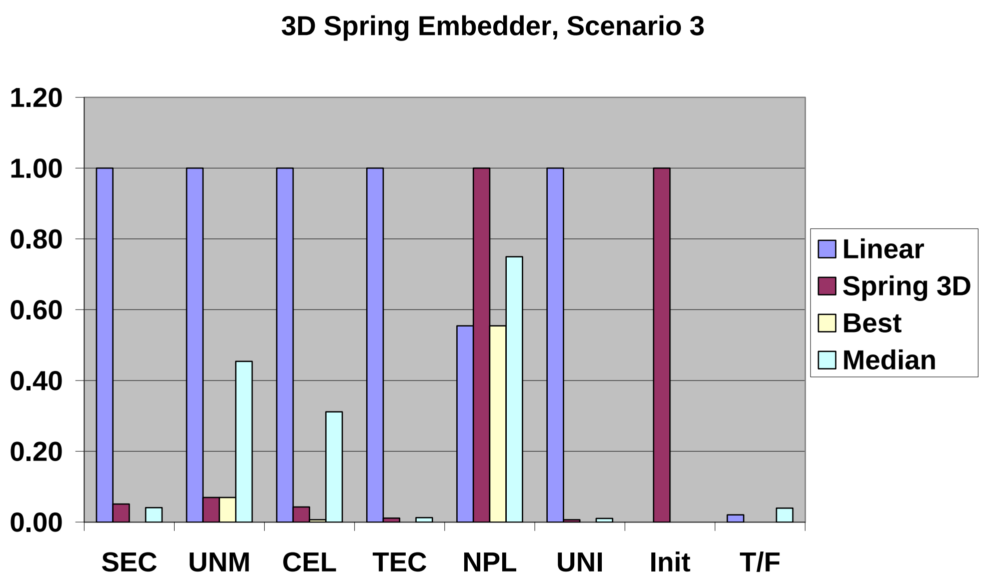


Figure 5.21: This figure shows 9 snapshots of a 101 frame animation. The animation is the result of applying the *3D Spring Embedder* method to scenario 3.

numerical results of the animation in regard to our measures are summarized in the following table:

Method	SEC	UNM	CEL	TEC	NPL	UNI	Init	T/F
Spring 3D	2	2.753E+07	4,887	45	170,902	154	62,450	0.0
Linear	40	3.962E+08	114,892	4,055	94,772	23,519	0	0.6
Best	0	2.753E+07	720	0	94,772	0	0	0
Median	2	1.799E+08	35,772	51	128,102	245	0	1.1



The results for scenario 3 are consistent with the remarks for the previous scenarios.

Scenario 4

Figure 5.22 shows the result of applying the *3D Spring Embedder* method to scenario 4. The

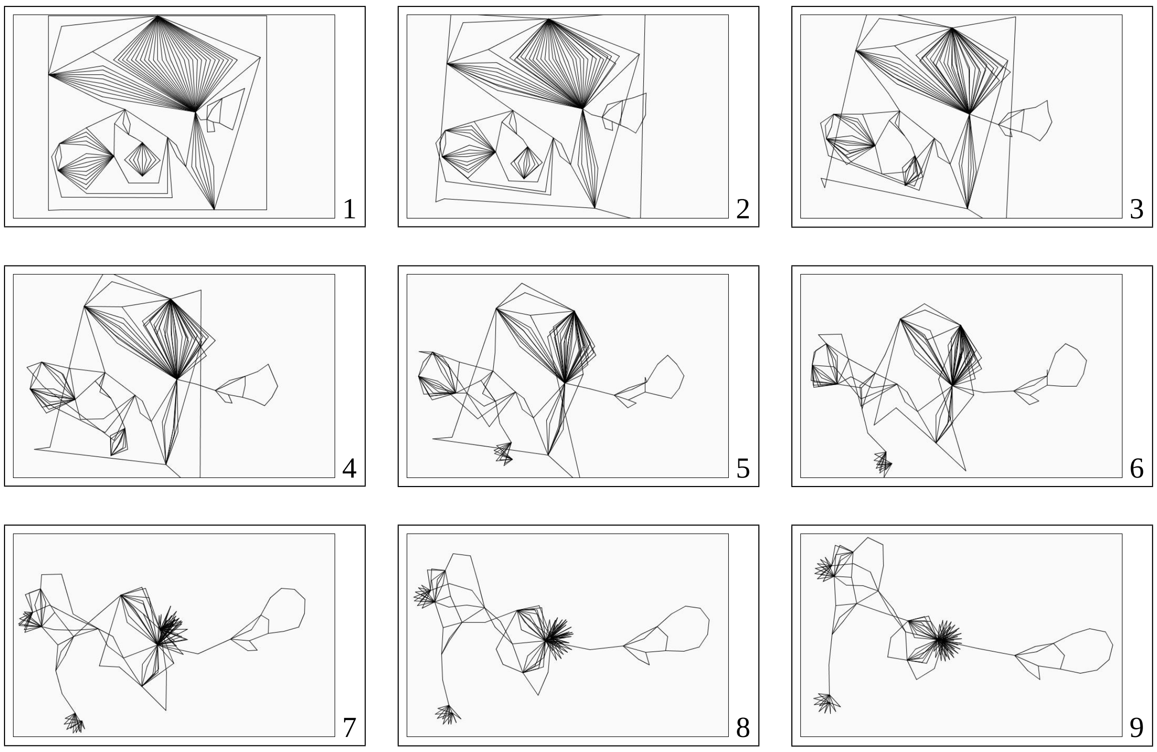
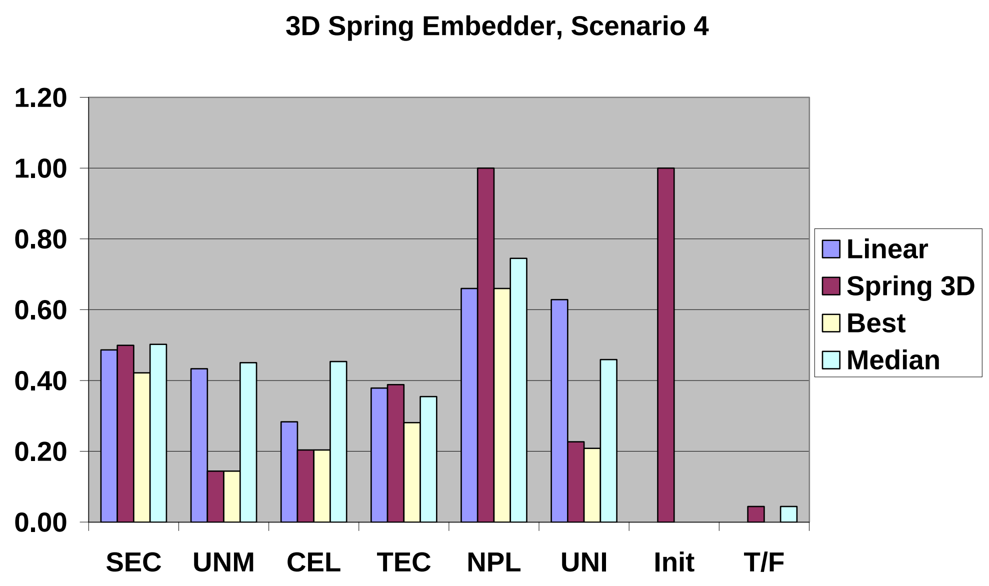


Figure 5.22: This figure shows 9 snapshots of a 101 frame animation. The animation is the result of applying the *3D Spring Embedder* method to scenario 4.

numerical results of the animation in regard to our measures are summarized in the following table:

Method	SEC	UNM	CEL	TEC	NPL	UNI	Init	T/F
Spring 3D	195	9.562E+07	954,237	1,362	182,710	61	26,420	0.5
Linear	190	2.877E+08	1,326,461	1,328	120,561	169	0	0.0
Best	164	9.562E+07	954,237	985	120,561	56	0	0
Median	196	2.991E+08	2,125,756	1,244	136,133	124	0	0.5



The results for scenario 4 are consistent with the remarks for the previous scenarios.

Scenario 5

Figure 5.23 shows the result of applying the *3D Spring Embedder* method to scenario 5. The

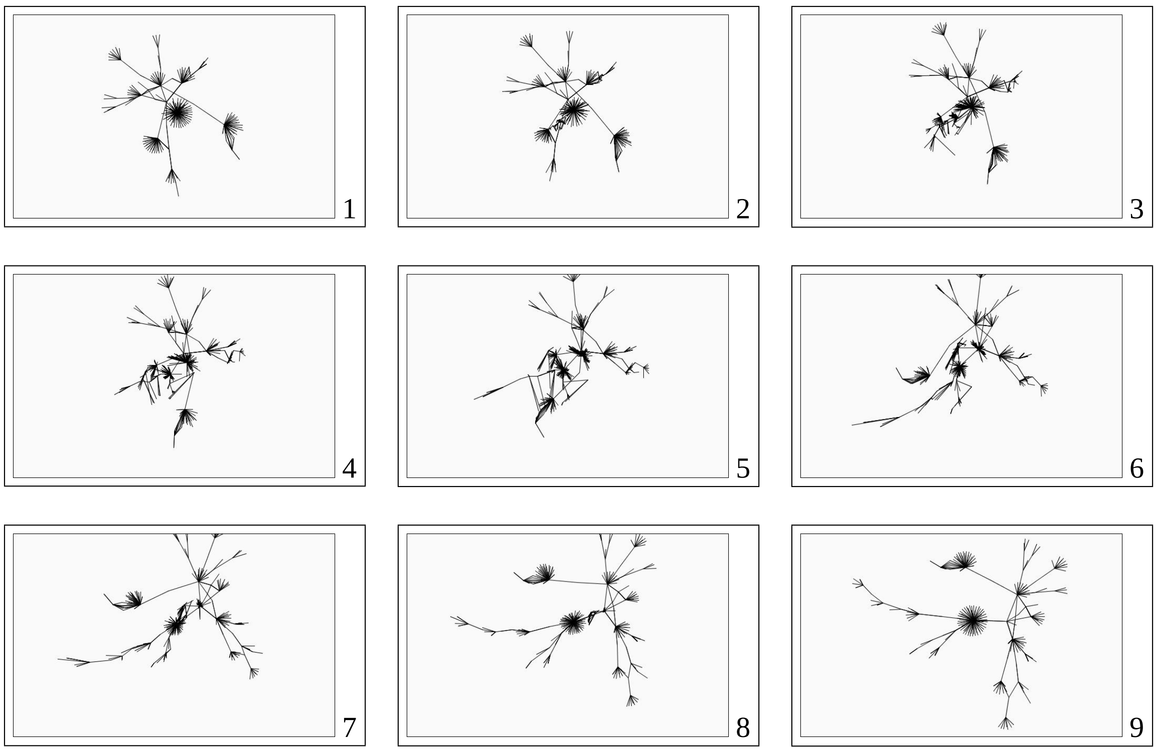
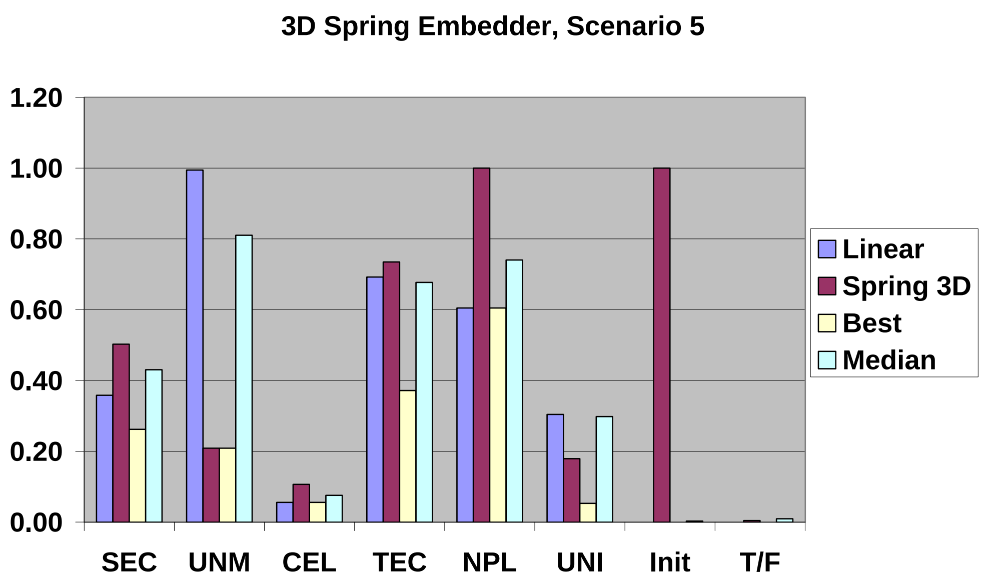


Figure 5.23: This figure shows 9 snapshots of a 101 frame animation. The animation is the result of applying the *3D Spring Embedder* method to scenario 5.

numerical results of the animation in regard to our measures are summarized in the following table:

Method	SEC	UNM	CEL	TEC	NPL	UNI	Init	T/F
Spring 3D	590	3.088E+09	4,591,048	16,692	987,276	163	21,470	0.5
Linear	420	1.470E+10	2,399,401	15,725	597,271	277	0	0.0
Best	308	3.088E+09	2,399,401	8,438	597,271	48	0	0
Median	505	1.198E+10	3,262,048	15,373	731,150	272	55	1.1



Scenario 5 is the first scenario where the *3D Spring Embedder* method shows results which are significantly worse than the results of the best method. Bad results occur in *static edge crossings*, *temporary edge crossings*, and *unnecessary node intersections*. We further notice that the actual animation is aesthetically not very pleasing. The emphasis in the *3D Spring Embedder* method to try to optimize *uniform node movement* and *constant edge length* prevents it from finding an aesthetically more pleasing animation with worse values for *uniform node movement* and *constant edge length* but better values for *static edge crossings*, *temporary edge crossings*, and *unnecessary node intersections*.

Scenario 6

Figure 5.24 shows the result of applying the *3D Spring Embedder* method to scenario 6. The

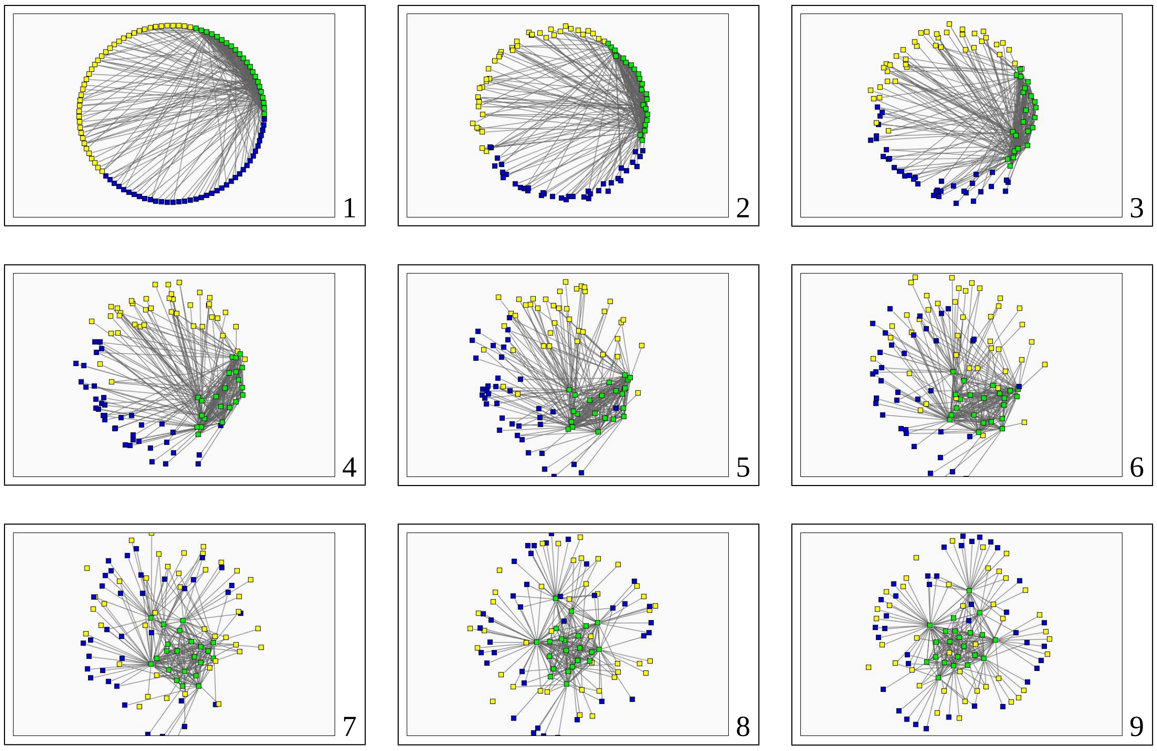
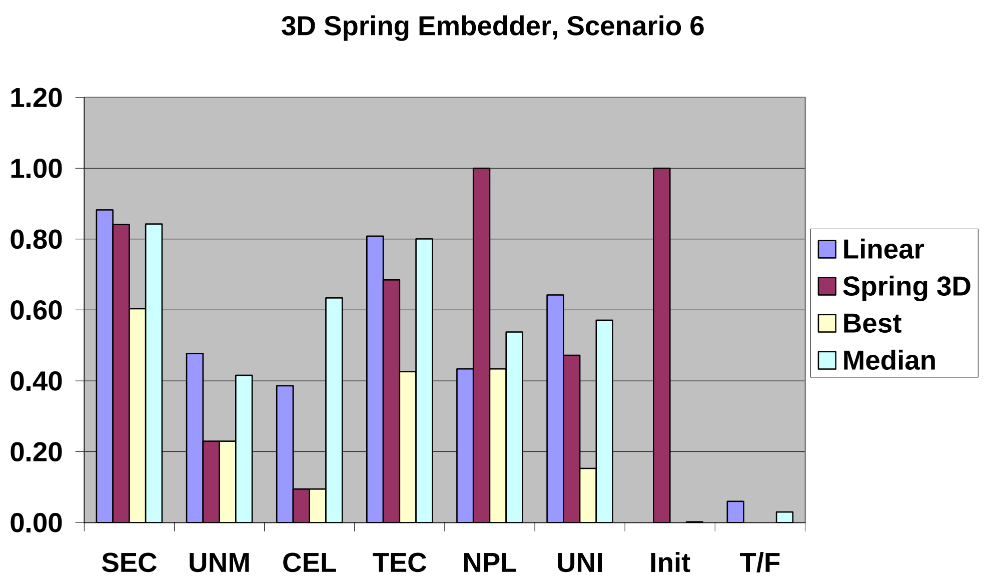


Figure 5.24: This figure shows 9 snapshots of a 101 frame animation. The animation is the result of applying the *3D Spring Embedder* method to scenario 6.

numerical results of the animation in regard to our measures are summarized in the following table:

Method	SEC	UNM	CEL	TEC	NPL	UNI	Init	T/F
Spring 3D	6,765	1.147E+08	1,020,039	13,348	170,545	136	12,640	0.0
Linear	7,095	2.385E+08	4,174,533	15,762	73,957	185	0	0.5
Best	4,853	1.147E+08	1,020,039	8,301	73,957	44	0	0
Median	6,777	2.077E+08	6,856,478	15,610	91,672	165	25	0.2



The results for scenario 6 are consistent with the remarks for the previous scenarios.

Scenario 7

Figure 5.25 shows the result of applying the *3D Spring Embedder* method to scenario 7. The

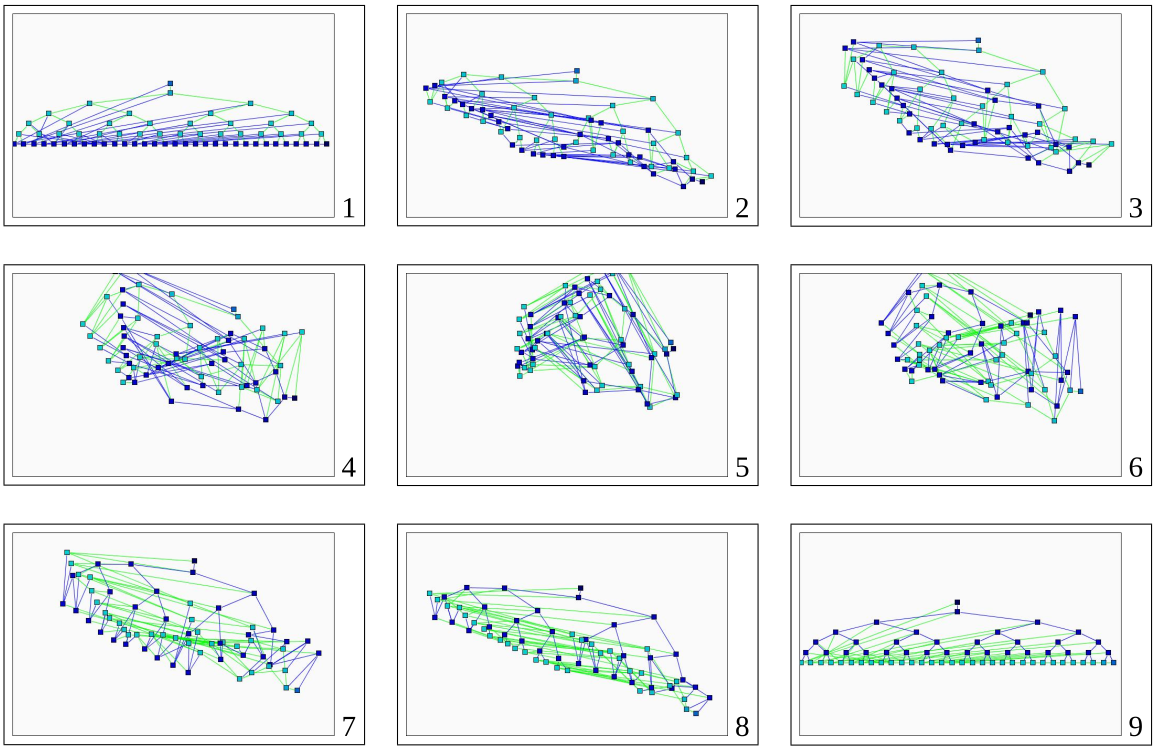
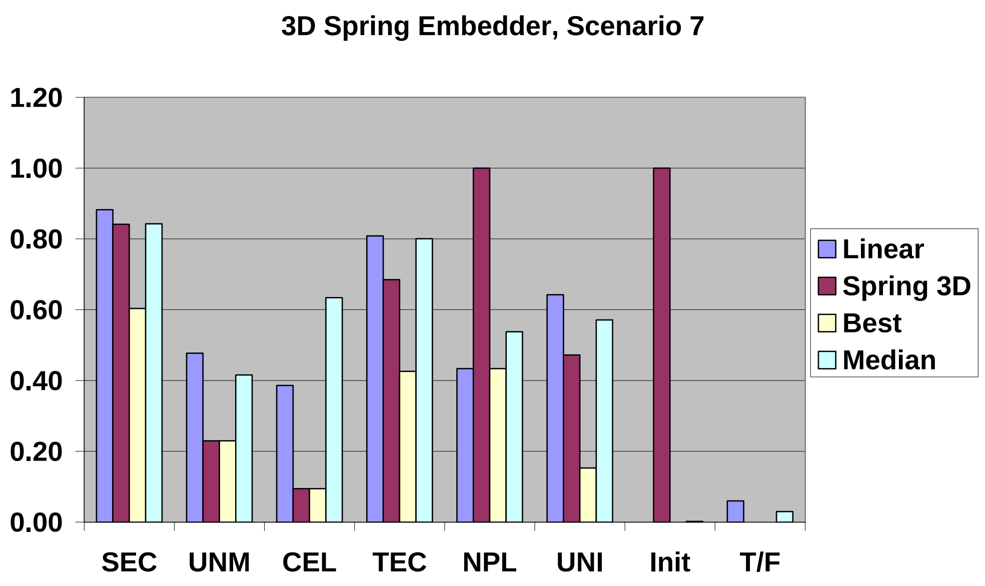


Figure 5.25: This figure shows 9 snapshots of a 101 frame animation. The animation is the result of applying the *3D Spring Embedder* method to scenario 7.

numerical results of the animation in regard to our measures are summarized in the following table:

Method	SEC	UNM	CEL	TEC	NPL	UNI	Init	T/F
Spring 3D	510	1.930E+07	71,857	4,345	65,639	146	10,550	2.3
Linear	1,011	4.858E+07	90,948	4,975	25,170	2,692	0	0.0
Best	454	1.841E+07	71,857	2,905	25,170	77	0	0
Median	696	4.262E+07	816,052	4,631	45,283	198	25	1.1



The *3D Spring Embedder* method is not able to find a good animation strategy in scenario 7. As per definition, the *3D Spring Embedder* method optimizes *uniform node movement* and *constant edge length*. However, the resulting animation has bad values for *static edge length*, *temporary edge length*, and *unnecessary node intersections*, and a particular bad value for *node path length*. The animation itself looks very inconsistent with many abrupt and large changes of direction at the virtual key frames.

Scenario 8

Figure 5.26 shows the result of applying the *3D Spring Embedder* method to scenario 8. The

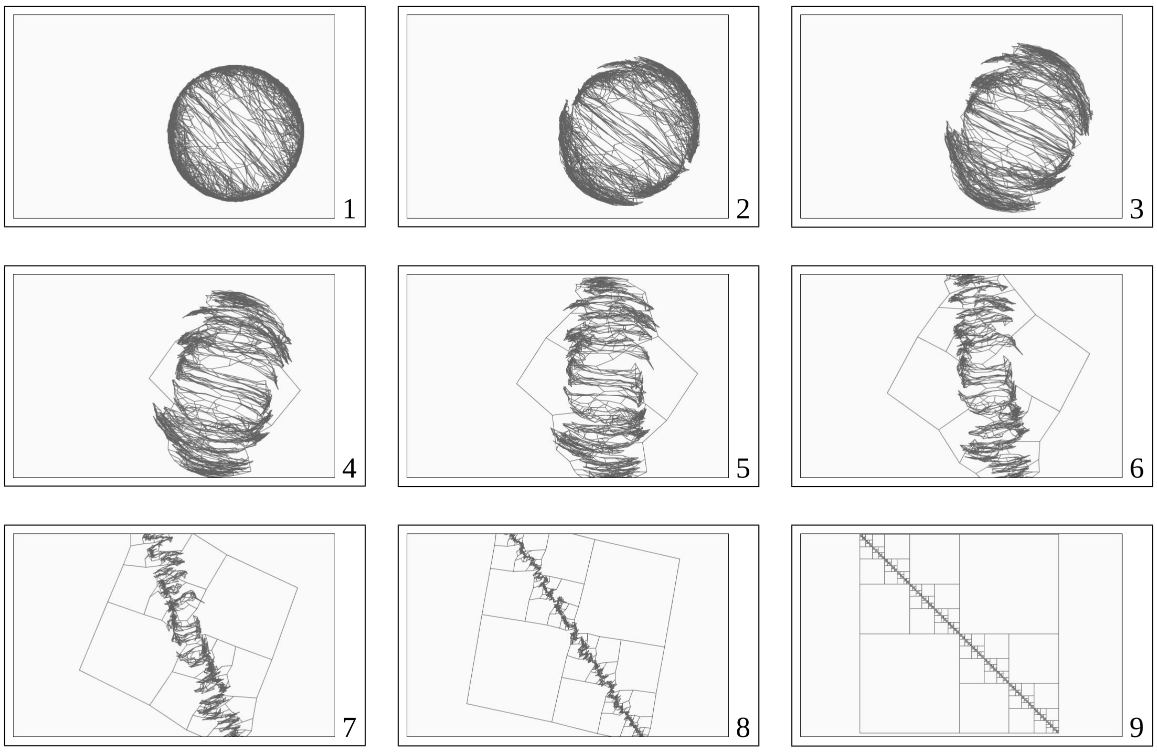
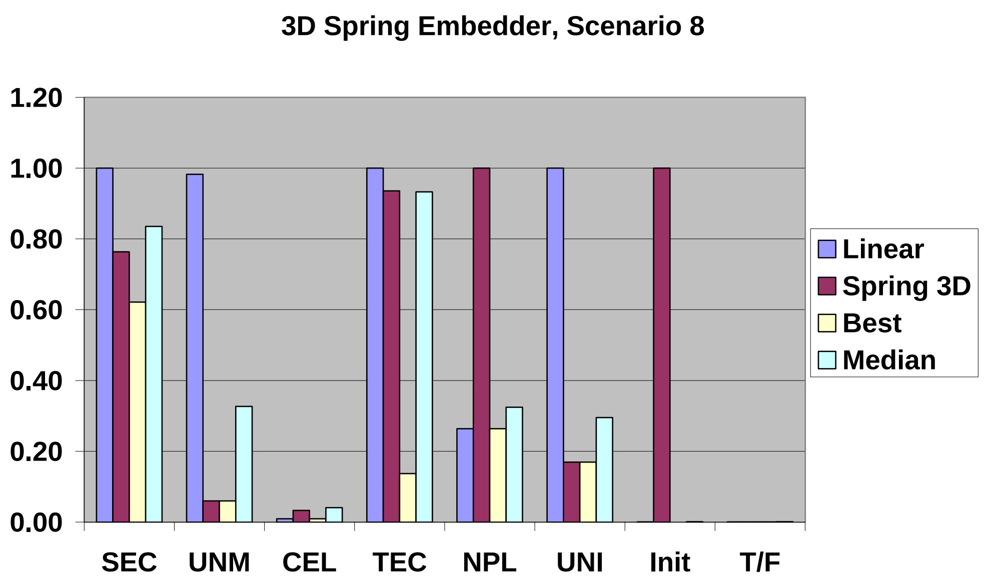


Figure 5.26: This figure shows 9 snapshots of a 101 frame animation. The animation is the result of applying the *3D Spring Embedder* method to scenario 8.

numerical results of the animation in regard to our measures are summarized in the following table:

Method	SEC	UNM	CEL	TEC	NPL	UNI	Init	T/F
Spring 3D	35,265	1.718E+08	14,816	1,977,113	1,100,867	1,256,611	87,500	5.0
Linear	46,185	2.829E+09	4,138	2,112,739	290,510	7,427,591	50	7.2
Best	28,706	1.718E+08	4,138	289,169	290,510	1,256,611	0	4
Median	38,574	9.407E+08	18,486	1,971,003	357,359	2,192,830	80	13.7



The results for scenario 8 are consistent with the remarks for the previous scenarios.

5.2.4 Summary

The *3D Spring Embedder* method produces good animations for most scenarios. Like the *Force Directed* method, it does not quite reach the same quality as specialized methods for certain specially structured scenarios, but still tends to produce good animations in these cases. The quality of the animations generated by the *3D Spring Embedder* method generally tends to be slightly less than for the *Force Directed* method. The *3D Spring Embedder* method is also more likely to produce perceptually worse animations than the *Force Directed* method, as for example in scenario 7 on page 198. In this scenario the *3D Spring Embedder* method is not able to find a good local optimum for the animation framework. The resulting animation is quite inconsistent and unstructured.

The *3D Spring Embedder* methods has several advantages over the *Force Directed* method, however. As the whole animation is computed before the first frame is displayed, the *3D Spring Embedder* methods can render each frame in linear time, provide random access to any frame, reverse the direction of the animation and allows efficient control over the speed of the animation. The biggest drawback of the *3D Spring Embedder* method is its high initialization cost. Although linear in complexity, a high constant factor almost always results in a noticeable gap before the animation starts on currently available computer systems. This gap can extend to several minutes for large graphs in certain scenarios. On current computer systems, the use of the *3D Spring Embedder* method is therefore limited to medium sized scenarios, non-interactive systems, or applications where the computational cost for computing the changes to the graph drawing outweigh the cost of initializing the animation.

CHAPTER 6

Linear Regression Analysis

Correlation analysis is a well known statistical method to determine dependencies between components of data samples [15, pp 800]. In the case of linear correlation, we assume our data set D has two components X and Y which have a linear dependency. This means, we assume that we can express one component as a linear combination of the other component plus some small error ϵ: $\exists a, b \in \mathbb{R} : \forall d \in D : d_y = a + bd_x + \epsilon_x$. Given such a dependency, linear regression allows us to determine a and b by minimizing the expression:

$$\sum_{i=1}^{n} (y_i - (a + bx_i))^2$$

Figure 6.1 shows an example for the case $X, Y \subseteq \mathbb{R}$. The dots represent the data samples. The diagonal line represents the affine linear function which minimizes the squared Euclidean distance to the data points. Not all data samples can be described by an affine linear transformation. It

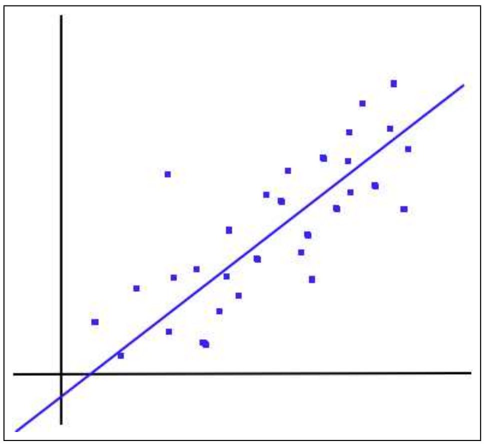


Figure 6.1: Example for one dimensional linear regression.

is therefore desirable to be able to determine how well the result of a linear regression analysis describes the actual data. A good and efficiently computable indicator for the “goodness-of-fit” of the computed affine linear transformation is the square of the correlation coefficient, the *coefficient of determination*. The coefficient of determination is usually denoted by r^2 and it can assume values between zero and one. A value of one indicates a perfect fit of the affine linear transformation. A value of zero indicates no linear dependency between the data samples. The value of r^2 can be interpreted as the percentage by which the prediction error is reduced when using the computed affine linear transformation compared to using the null assumption, that is the mean over all data samples. For a more detailed discussion on the coefficient of determination compare also [15, pp 800] and [81, pp 590].

In section 2.4 we discussed how the human brain is genetically predisposed for perceiving certain kinds of “natural” motions. Among these motions are the movements of three dimensional rigid objects in space. This fact is hardly surprising, as a fast and efficient perception of those movements is essential for us to successfully interact with our environment in every day life. Even when we project the movement of a three dimensional rigid object onto a plane we can still perceive it with the same efficiency. For example it is not more difficult for us to follow actions on a TV screen or with only one eye than when we have full stereo vision. If we project the motion of a three dimensional rigid object onto a plane, then the projected motion can be described as a combination of two dimensional scaling, rotating, translating, flipping and shear operations. For example an object moving away from us can be described by scaling the projection. Rotations can be described by rotation in the plane or scaling along one axis. The combinations of scaling, rotating, translating, flip, and shear form the set of two dimensional *affine linear transformations*. Given the initial and target positions of the nodes of a graph we can use linear regression to determine the best affine linear transformation to describe this motion. We can then break this transformation down into the basic operations and interpolate them.

6.1 Linear Regression Analysis Method

6.1.1 Linear Regression Analysis of Node Positions

Given a graph G and the coordinates (x_v, y_v) for each node $v \in G$, an affine linear transformation of the node coordinates has the general form:

$$f(v) = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix} \begin{pmatrix} x_v \\ y_v \end{pmatrix} + \begin{pmatrix} a_{13} \\ a_{23} \end{pmatrix}$$

In order to determine the linear function which transforms the graph as close as possible to the target frame, we have to determine the a_{ij} which minimize the distance between the positions in the target frame and the positions after the linear transformation. We use the average error between the individual node distances as a good approach in respect to the goal of preserving the mental map; that is we minimize the sum of the squared Euclidean distances. Given (x'_v, y'_v) as the coordinates of a node v in the target drawing, we aim to minimize

$$\sum_{v \in G} |f(v) - (x'_v, y'_v)|$$

with $||$ being the squared Euclidean norm, that is

$$\sum_{v \in G} \left(a_{11}x_v + a_{12}y_v + a_{13} - x'_v\right)^2 + \left(a_{21}x_v + a_{22}y_v + a_{23} - y'_v\right)^2 \tag{6.1}$$

Since expression (6.1) is always non-negative, minimizing this quadratic function can be done by deriving it with respect to the a_{ij} and setting the resulting expressions to equal 0. We solve the system of equations for the a_{ij}. Assuming at least 3 non co-linear pairs of node coordinates, this gives us a unique solution for each a_{ij}:

Let

$$\begin{aligned} denom \;=\; & \left(\sum_{v \in G} x_v^2\right)\left(n\sum_{v \in G} y_v^2 - \left(\sum_{v \in G} y_v\right)^2\right) - n\left(\sum_{v \in G} x_v y_v\right)^2 + \left(\sum_{v \in G} x_v\right)\cdots \\ & \cdots\left(2\left(\sum_{v \in G} y_v\right)\sum_{v \in G} x_v y_v - \left(\sum_{v \in G} x_v\right)\sum_{v \in G} y_v^2\right). \end{aligned}$$

Then

$$
\begin{aligned}
a_{11} &= -\Bigg(\frac{\left(\sum_{v\in G} x_v y_v\right)\left(n\sum_{v\in G} x'_v y_v - \left(\sum_{v\in G} x'_v\right)\sum_{v\in G} y_v\right)}{denom} + \\
&\quad \frac{\left(\sum_{v\in G} x'_v x_v\right)\left(\left(\sum_{v\in G} y_v\right)^2 - n\sum_{v\in G} y_v^2\right)}{denom} + \\
&\quad \frac{\left(\sum_{v\in G} x_v\right)\left(\left(\sum_{v\in G} x'_v\right)\sum_{v\in G} y_v^2 - \left(\sum_{v\in G} y_v\right)\sum_{v\in G} x'_v y_v\right)}{denom} \Bigg), \\
a_{12} &= \frac{\left(\sum_{v\in G} x_v^2\right)\left(n\sum_{v\in G} x'_v y_v - \left(\sum_{v\in G} x'_v\right)\sum_{v\in G} y_v\right)}{denom} + \\
&\quad \frac{\left(\sum_{v\in G} x_v y_v\right)\left(\left(\sum_{v\in G} x'_v\right)\sum_{v\in G} x_v - n\sum_{v\in G} x'_v x_v\right)}{denom} + \\
&\quad \frac{\left(\sum_{v\in G} x_v\right)\left(\left(\sum_{v\in G} y_v\right)\sum_{v\in G} x'_v x_v - \left(\sum_{v\in G} x_v\right)\sum_{v\in G} x'_v y_v\right)}{denom}, \\
a_{13} &= \frac{\left(\sum_{v\in G} x_v^2\right)\left(\left(\sum_{v\in G} x'_v\right)\sum_{v\in G} y_v^2 - \left(\sum_{v\in G} y_v\right)\sum_{v\in G} x'_v y_v\right)}{denom} - \\
&\quad \frac{\left(\sum_{v\in G} x'_v\right)\left(\sum_{v\in G} x_v y_v\right)^2}{denom} + \\
&\quad \frac{\left(\sum_{v\in G} x_v y_v\right)\left(\left(\sum_{v\in G} y_v\right)\sum_{v\in G} x'_v x_v + \left(\sum_{v\in G} x_v\right)\sum_{v\in G} x'_v y_v\right)}{denom} - \\
&\quad \frac{\left(\sum_{v\in G} x_v\right)\left(\sum_{v\in G} y_v^2\right)\sum_{v\in G} x'_v x_v}{denom}.
\end{aligned}
\tag{6.2}
$$

We can obtain a_{21}, a_{22}, and a_{23} from the solutions for a_{11}, a_{12}, and a_{13} respectively by replacing x'_v with y'_v.

Interpolating an affine linear transformation

We have so far computed the best affine linear transformation between the initial and target drawing of the graph. As the identity function describes the node positions in the initial drawing, an animation can be generated by interpolating between the identity matrix and the matrix resulting from the linear regression analysis. The easiest approach would be to do a linear interpolation of each matrix entry from the identity matrix to the computed matrix. However, the result would not produce the desired effect of being perceived as the movement of a rigid object in all cases. This can easily be seen in the case of a 180^o rotation of the graph. The computed transformation matrix in this case would be

$$tm = \begin{pmatrix} -1 & 0 \\ 0 & -1 \end{pmatrix}$$

interpolating the entries from the identity matrix

$$\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$$

describing the initial drawing of the graph to tm produces the matrix

$$\begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix}$$

at some stage collapsing the graph to one point. This does not happen in a 180 degree rotation.

Shoemake and Duff [102] show that the rotation component of the transformation is the only part not compatible with linear entry interpolation and propose polar matrix decomposition as an efficient way to separate a maximal rotational part from the transformation. For a 2×2 matrix[1], M, the polar decomposition, $M = RS$, into a rotational matrix, R, and a non-rotational matrix, S, takes constant time: Given $M = \begin{pmatrix} a & b \\ c & d \end{pmatrix}$ and $\sigma_M = sign\left(det\left(M\right)\right)$ then

$$R = M + \sigma_M \begin{pmatrix} d & -c \\ -b & a \end{pmatrix}$$

and

$$S = \begin{pmatrix} \sigma_M\left(ad - bc\right) + a^2 + c^2 & ab + cd \\ ab + cd & \sigma_M\left(ad - bc\right) + b^2 + d^2 \end{pmatrix}$$

We can now generate an animation by interpolating the rotation over the angle, and at the same time do a linear interpolation of the matrix entries of the non-rotational part.

Adjusting the center of rotation

The approach described above still has a small problem. The rotation that we compute is always performed around the point of origin. The resulting motion path is thus an arc which size depends on how far the graph lies from the origin. To avoid this behavior we modify our linear transformation before the decomposition to move the center of the graph to the origin at the beginning and back to the original position afterwards. This can easily be done by multiplying the corresponding

[1]For this purpose we can ignore the translation part

transformation matrices to each side of our matrix. The graph now rotates around its center. Several definitions of center are possible, for example the center of the bounding box, or the barycenter. In our experience the decision of which definition to use had little influence on the overall animation.

6.1.2 The Method

We can now define an animation method based on linear regression analysis. The following steps have to be performed:

1. Compute the center of the of the initial and target drawing of the graph.

2. Compute an affine linear transformation from the initial layout to the target layout and an affine linear transformation from the target layout to the initial layout. Note that the second transformation is generally not identical to the inverse of the first transformation. Figure 6.2 illustrates this fact. Drawing 6.2.1 can be transformed into a drawing which is very close to

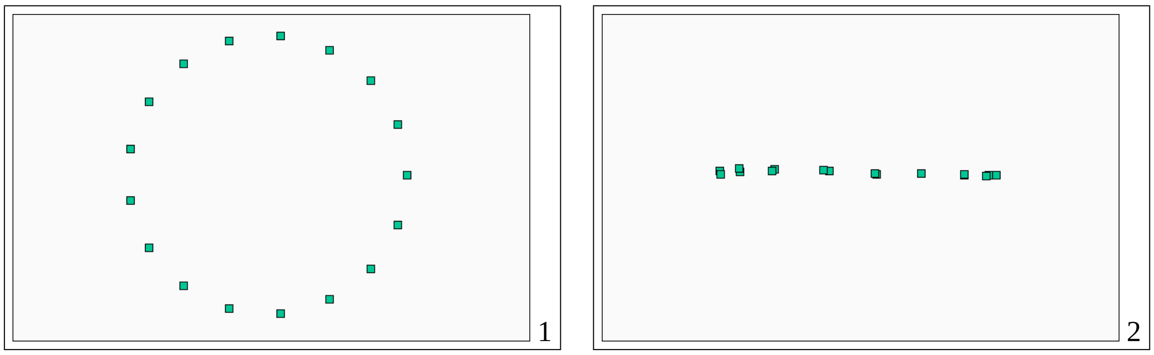


Figure 6.2: Example for a scenario in which the computed best affine linear transformation from the initial to the target drawing is not the inverse of the best affine linear transformation from the target drawing to the initial drawing.

   drawing 6.2.2 by scaling the graph along the y-axis. However, an affine linear transformation which transforms drawing 6.2.2 into a drawing which is close to drawing 6.2.1 does not exist.

3. Decompose both transformations using polar decomposition.

4. For each frame compose two transformations by interpolating the two rotation angles and the non-rotational matrices. The two transformations are then combined with a weight which corresponds to the progress of the animation. Given a graph $G = (V, E)$ and two affine linear transformations $A_1 = (\alpha_1, S_1, T_1)$ and $A_2 = (\alpha_2, S_2, T_2)$ from the initial to the target layout

and vice versa, with α_i being the rotational angles, S_i being a non rotational matrices, and T_i being the translation components of the transformations, the position of each node $v \in V$ at time t is given by:

$$\begin{aligned} P_v(t) &= (1-t)\, tT_1\mathcal{R}(t\alpha_1)(I + t(S_1 - I))\, P_v(0) + \cdots \\ &\quad \cdots t(1-t)\, T_2\mathcal{R}((1-t)\,\alpha_2)(I + (1-t)(S_2 - I))\, P_v(1) \end{aligned}$$

with $\mathcal{R}$ being a function transforming an angle into its corresponding rotational matrix and I being the identity matrix.

Variation: Adding visual clues The computed transformation can not only be used to compute the movements of the nodes. We can also use it to transform parts of the environment of the animation, for example the canvas. Transforming the canvas increases the impression of a moving rigid object. The example in figure 6.3 shows snapshots from an animation with added canvas transformation effects. Other visual effects are possible as well.

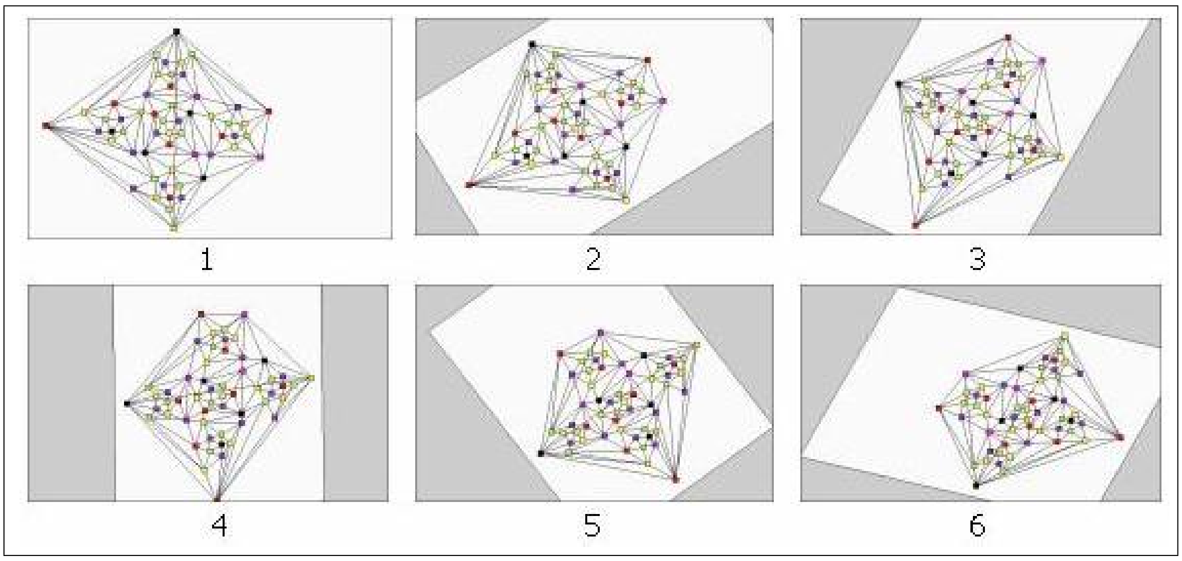


Figure 6.3: Adding visual effects can increase the impression of the graph moving as a rigid object in space.

6.1.3 Analysis

1. Init

   Given a graph $G = (U, V)$ and $n|V|$ the following steps have to be performed:

   - Determine center of the initial and target layout. This costs $O(n)$ time.
   - Compute transformations from the initial to the target layout and vice versa using linear regression. The matrix indices a_{ij} in equation (6.2) can be computed in one iteration over all nodes and therefore cost $O(n)$.
   - Extract the rotation angle using polar decomposition. This costs $O(1)$ time.

   This amounts to a total *Init* time of $O(n)$.

2. T/F

   The position of every node can be updated in constant time, resulting in a time complexity of $O(n)$ to compute a frame.

3. SEC, TEC, UNI

   As long as an animation is topology preserving, it achieves the optimal value zero for *static edge crossings*, *temporary edge crossings*, and *unnecessary node intersections*. The *Linear Regression Analysis* method computes an animation which is based on the combination of two affine linear transformations. Each of the two affine linear transformations is topology preserving in itself; with the exception of a possible flip. However, one transformation is applied to the initial positions of the nodes, and the other transformation is applied to the target positions of the nodes. As a consequence, the averaging combination of the two positions for each node is not necessarily topology preserving. The ability to break the graph topology is even necessary, as otherwise the *Linear Regression Analysis* method would not be able to create animations between arbitrary non-topology preserving drawings of graphs. For scenarios where the target drawing can be described sufficiently well by an affine linear transformation of the initial drawing, the *Linear Regression Analysis* method thus achieves close to optimal values for *static edge crossings*, *temporary edge crossings*, and *unnecessary node intersections*. If no such affine linear dependency exists, however, arbitrarily bad values for these measures can occur. Figure 6.4 shows an example. The right subgraph in the initial drawing 6.4.1 moves to the left, while the left subgraph remains at its original position. The

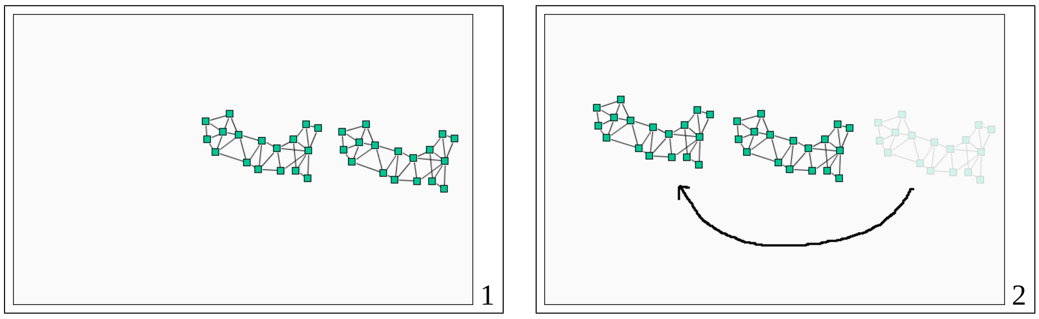


Figure 6.4: Example for a scenario where the *Linear Regression Analysis* method achieves bad values for most measures.

Linear Regression Analysis method finds the following affine linear transformation for this scenario:

Transformation	Amount
Scale along x-Axis	0.631
Scale along y-Axis	1.203
Rotation	-167^o
Skew	-42^o
Translation	$(227.78, 20)$
Flip	True

The r^2 value of the computed transformation is only 35.44 %, indicating a low probability for an affine linear dependency between the initial drawing and the target drawing. Also, intuitively, the decomposed transformation seems quite counter-intuitive. We would expect the right subgraph to move uniformly, maybe on an arc, to the left. A *flip* and the high value for *skew* in the transformation are not necessary for such a movement. The corresponding animation, which is shown in figure 6.5, confirms our apprehensions. The animation achieves unnecessarily bad results for almost all measures.

4. CEL, UNM

As for *static edge crossings*, *temporary edge crossings*, and *unnecessary node intersections* the values for *constant edge length* and *uniform node movement* depend strongly on the degree of affine linear dependency between the initial and the target drawing. However, in this case the dependency is not quite as straightforward. The determining factor for *constant edge length* and *uniform node movement* is whether the computed *skew* factor is valid or not. For

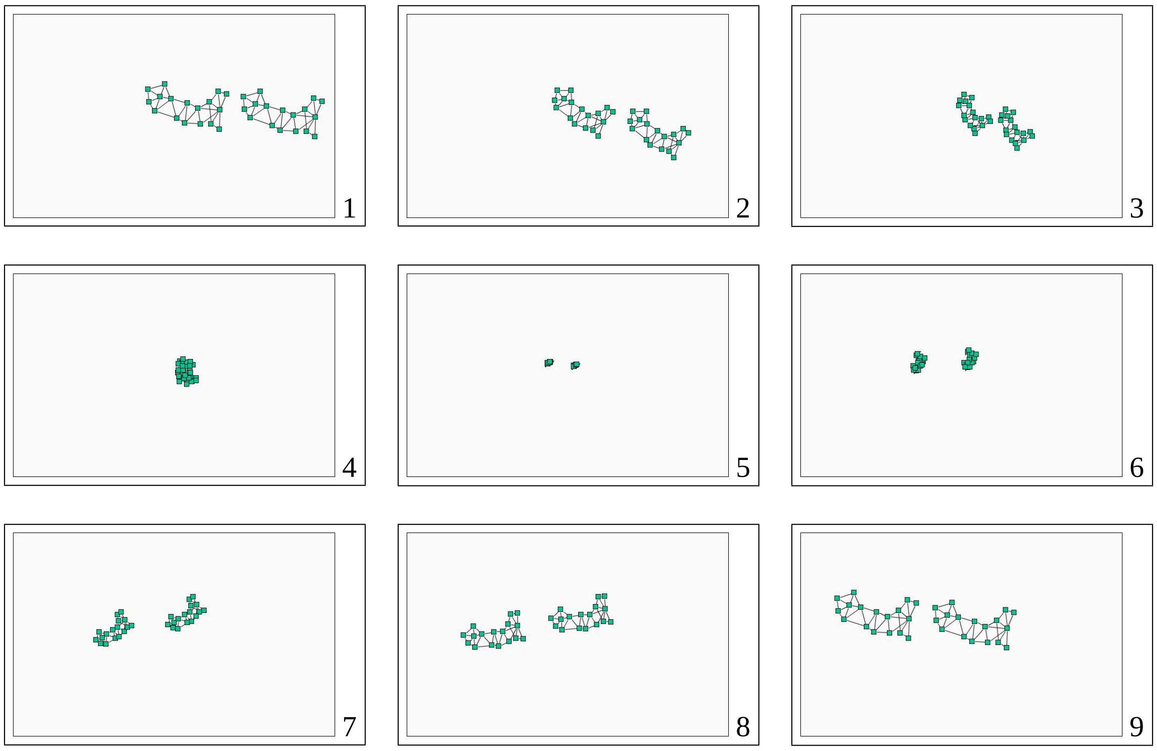


Figure 6.5: This figure shows 9 snapshots of a 101 frame animation of the scenario displayed in figure 6.4. The animation is the result of applying the *Linear Regression Analysis* method. This particular scenario cannot be described well by an affine linear transformation, and the animation achieves bad values for most measures.

transformations which contain a natural *skew* component, animating a the *skew* does not affect *constant edge length* or *uniform node movement*. However, if the *Linear Regression Analysis* method wrongly identifies a non-existent *skew* component, as for example in figure 6.5, *constant edge length* and *uniform node movement* can become arbitrarily bad.

5. Smoothness

   The path of each node is given by a combination of linear transformations, and is therefore smooth. In multiple key frame scenarios, a non-smooth direction change is expected at every key frame.

6.1.4 Empirical Evaluation

Scenario 1

Figure 6.6 shows the result of applying the *Linear Regression Analysis* method to scenario 1. The

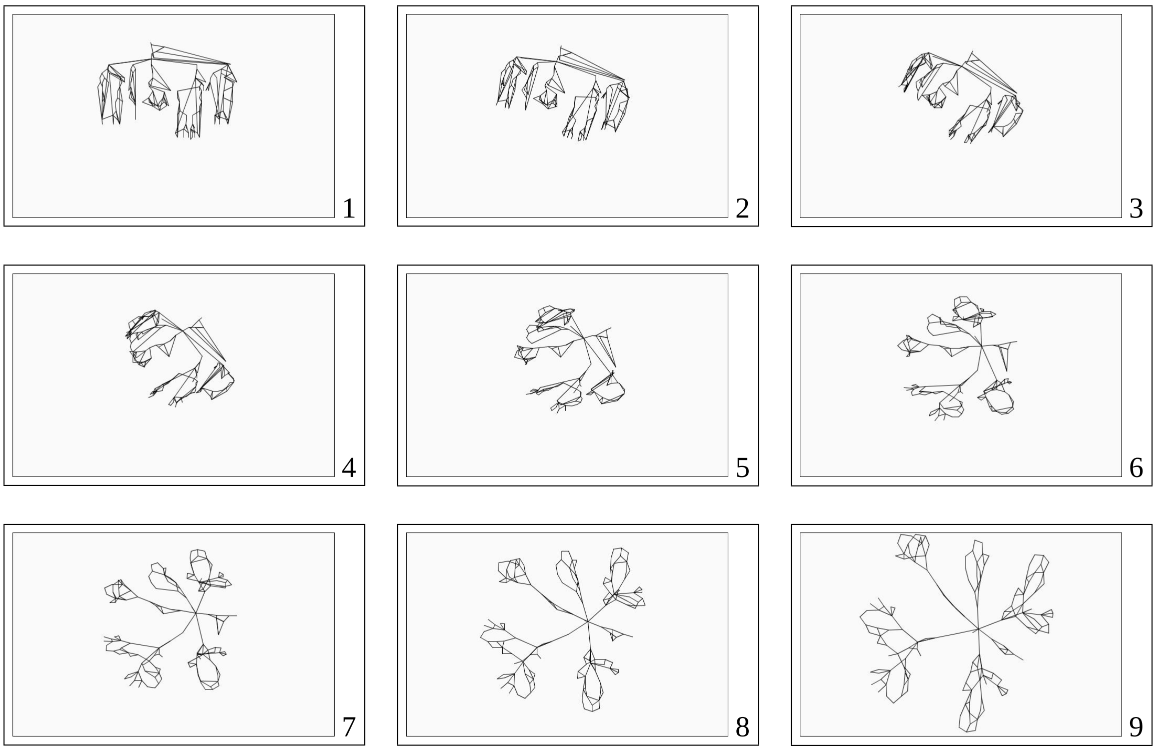
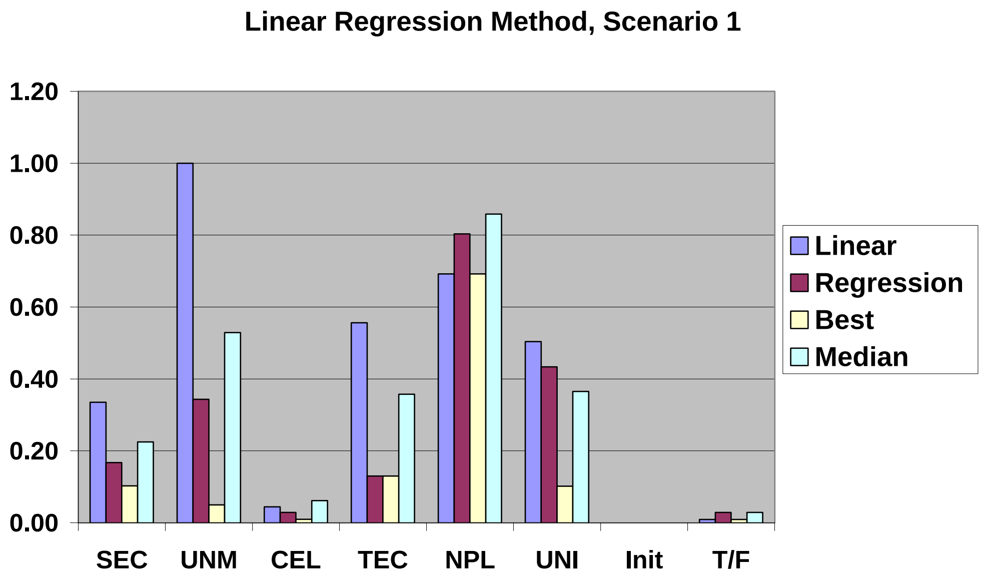


Figure 6.6: This figure shows 9 snapshots of a 101 frame animation. The animation is the result of applying the *Linear Regression Analysis* method to scenario 1.

numerical results of the animation in regard to our measures are summarized in the following table:

Method	SEC	UNM	CEL	TEC	NPL	UNI	Init	T/F
Regression	65	1.722E+09	1,203,360	490	404,066	111	0	1.6
Linear	130	5.020E+09	1,843,276	2,095	347,983	129	0	0.5
Best	40	2.484E+08	402,170	490	347,983	26	0	0
Median	87	2.656E+09	2,577,963	1,346	431,802	94	0	1.6



The measured goodness-of-fit value in scenario 1 is $r^2 = 46\%$ for the linear transformation from the initial to the target drawing, and $r^2 = 59\%$ for the reverse transformation.

The *Linear Regression* method achieves better values than the *Direct Linear Interpolation* method regarding all measures but *node path length*. The *Linear Regression* method also achieves better results than the median in all measures but *unnecessary node intersections*. The animation itself appears aesthetically pleasing.

We notice a significant difference between the r^2 values of the transformation from the initial drawing to the target drawing and the transformation from the target drawing to the initial drawing. As discussed before, the two transformations are not necessarily inverse to each other, and also not necessarily are able to describe the transformation with the same goodness of fit. This can result in significantly different r^2 values for the two transformations.

Scenario 1b

Figure 6.7 shows the result of applying the *Linear Regression Analysis* method to scenario 1b. The

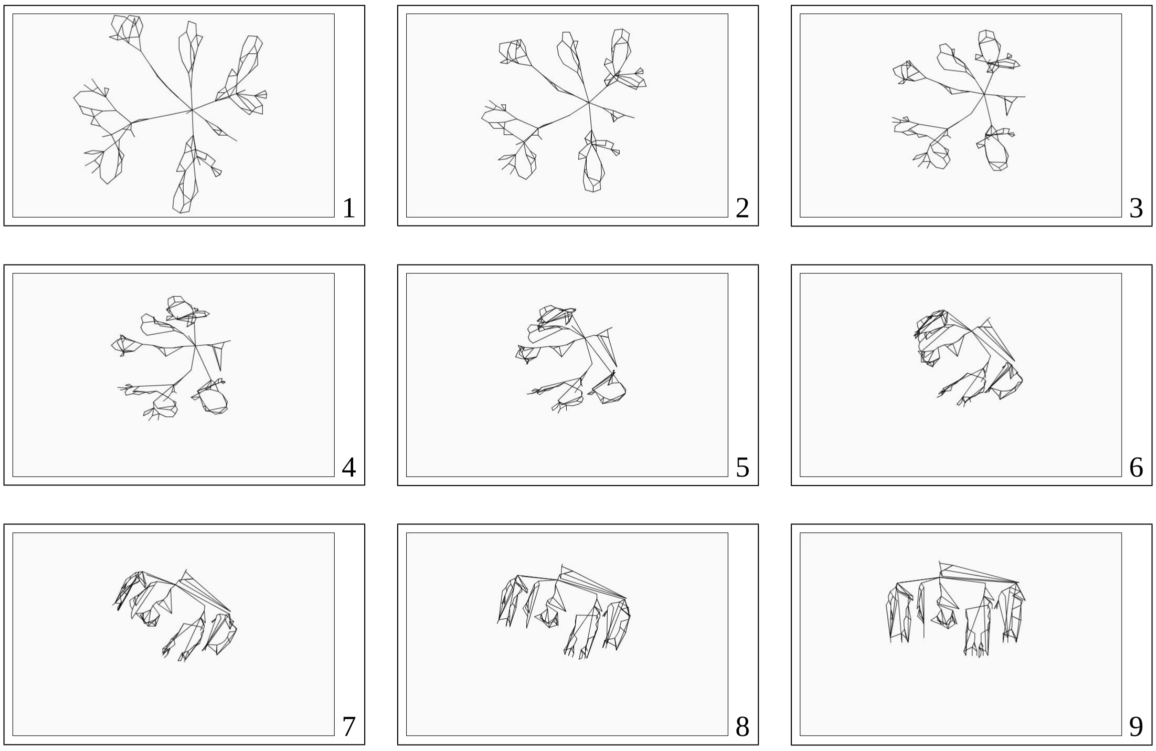
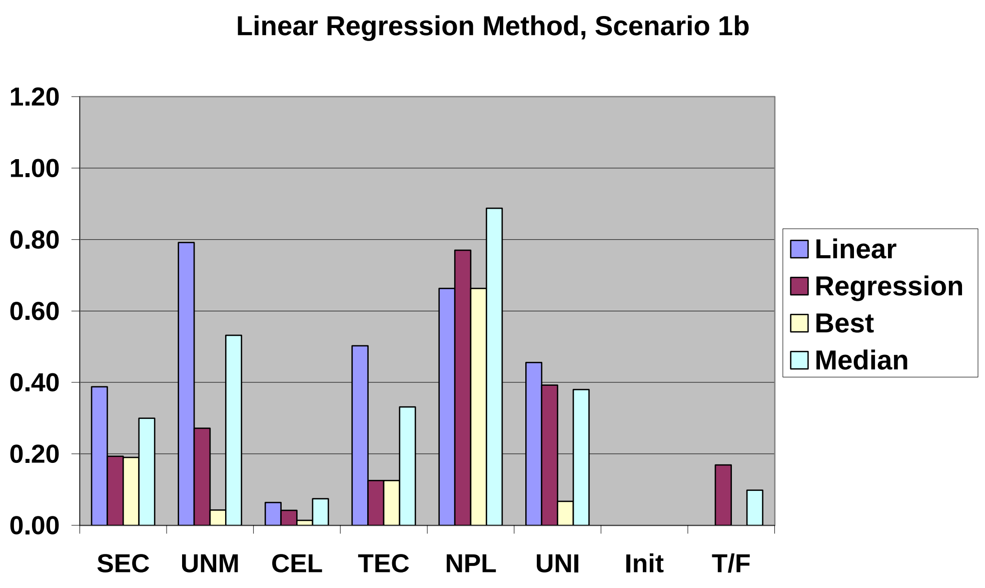


Figure 6.7: This figure shows 9 snapshots of a 101 frame animation. The animation is the result of applying the *Linear Regression Analysis* method to scenario 1b.

numerical results of the animation in regard to our measures are summarized in the following table:

Method	SEC	UNM	CEL	TEC	NPL	UNI	Init	T/F
Regression	65	1.722E+09	1,203,360	533	404,055	111	0	8.8
Linear	130	5.020E+09	1,843,276	2,138	347,983	129	0	0.0
Best	64	2.697E+08	395,880	533	347,983	19	0	0
Median	100	3.371E+09	2,146,880	1,409	465,534	108	0	5.1



The measured goodness-of-fit value in scenario 1b is $r^2 = 59\%$ for the linear transformation from the initial to the target drawing, and $r^2 = 46\%$ for the reverse transformation. The *Linear Regression* method is theoretically symmetric for exchanged initial and target drawings. The small deviations in the measures are due to numerical inaccuracies.

Scenario 2

Figure 6.8 shows the result of applying the *Linear Regression Analysis* method to scenario 2. The

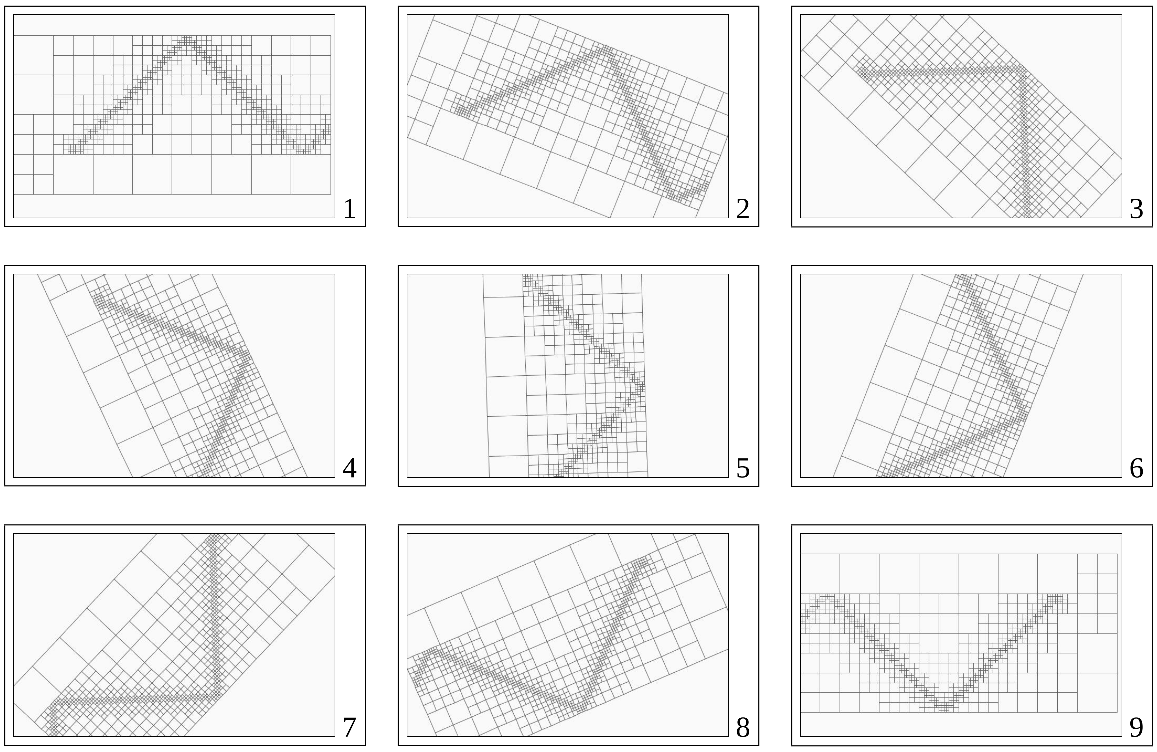
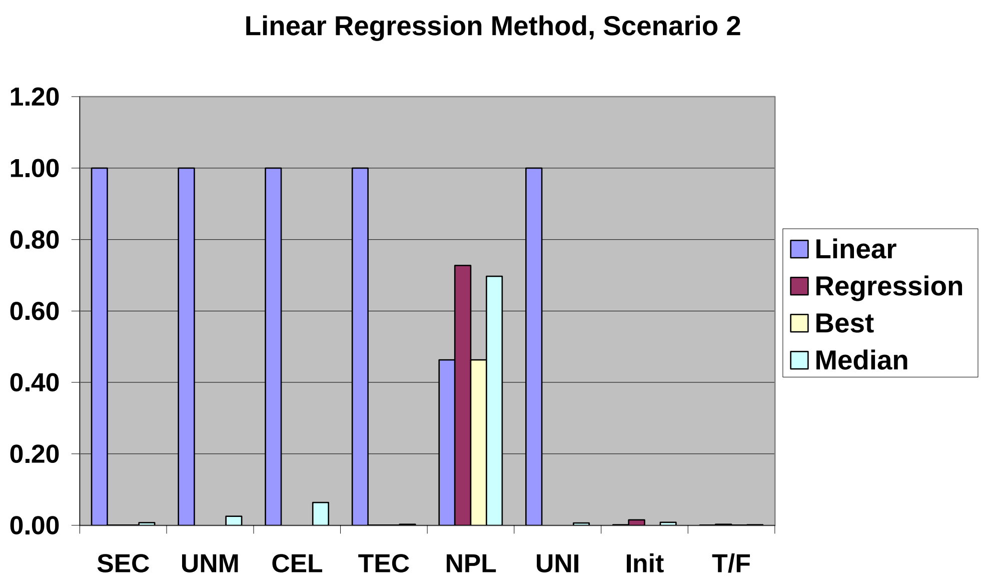


Figure 6.8: This figure shows 9 snapshots of a 101 frame animation. The animation is the result of applying the *Linear Regression Analysis* method to scenario 2.

numerical results of the animation in regard to our measures are summarized in the following table:

Method	SEC	UNM	CEL	TEC	NPL	UNI	Init	T/F
Regression	6	1.066E-20	0	578	691,249	0	490	7.7
Linear	12,891	1.675E+10	212,992	1,301,941	440,110	3,813,032	50	1.6
Best	4	1.066E-20	0	374	440,110	0	0	1
Median	92	4.259E+08	13,608	3,468	662,794	24,579	270	4.7



The measured goodness-of-fit value in scenario 2 is $r^2 = 100\%$ for the linear transformation from the initial to the target drawing, and $r^2 = 100\%$ for the reverse transformation. The goodness-of-fit value of $r^2 = 100\%$ shows us that the transformation from the initial drawing to the target drawing can be perfectly described by an affine linear transformation. The *Linear Regression* method consequently achieves close to optimal values for all measures but *node path length*. Although the *Linear Regression* method is able to describe the transformation perfectly, numerical inaccuracies cause small deviations form the theoretical optimal values.

Scenario 2b

Figure 6.9 shows the result of applying the *Linear Regression Analysis* method to scenario 2b. The

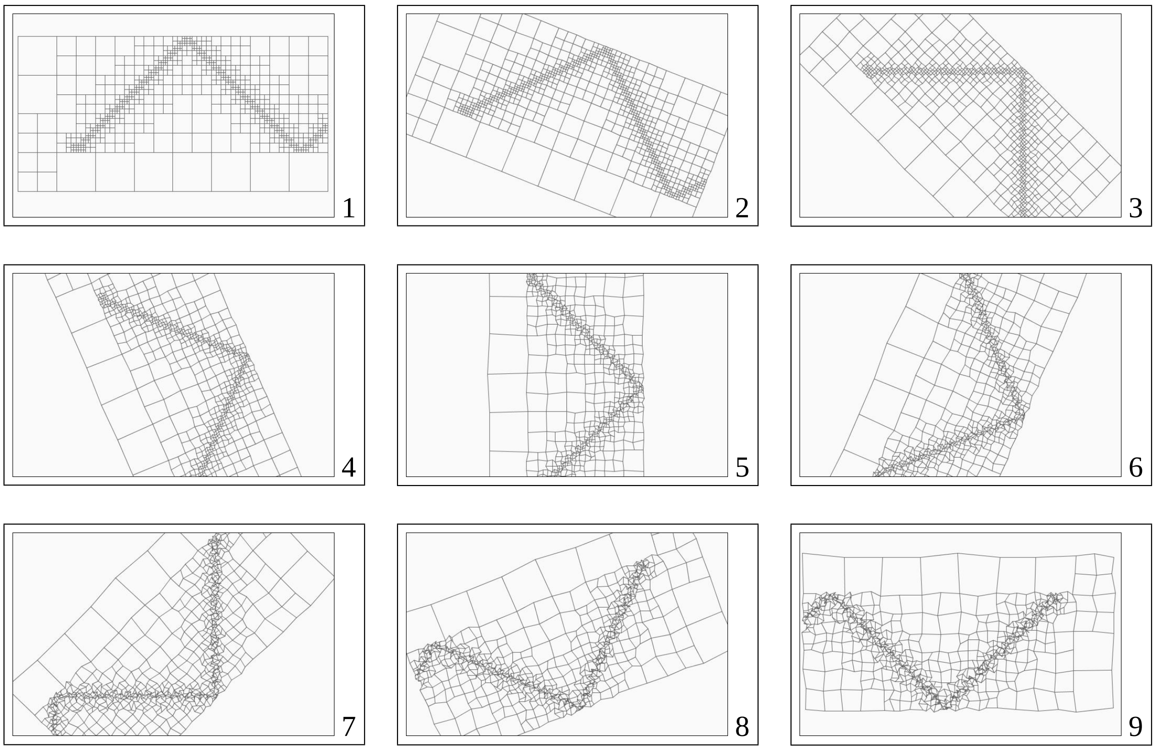
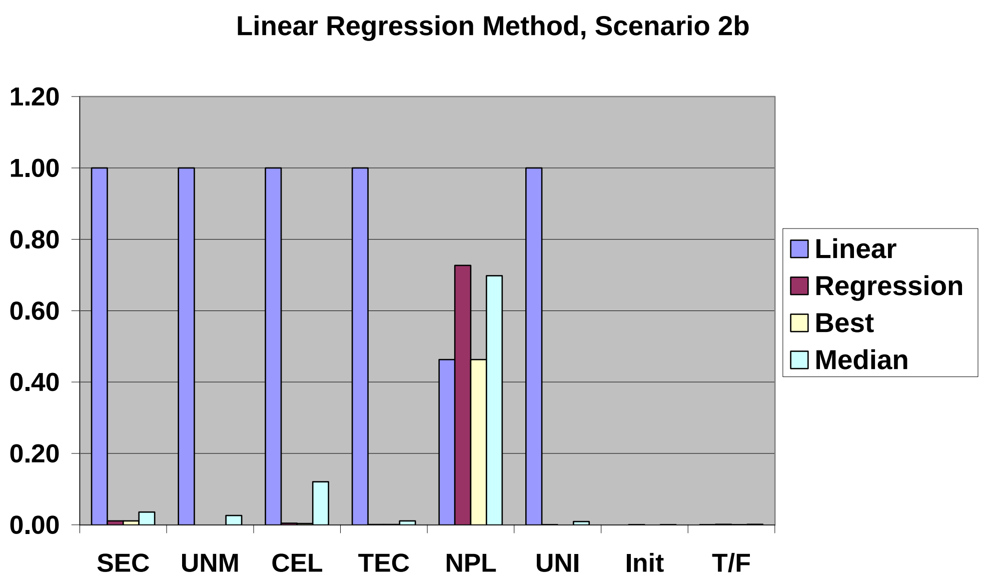


Figure 6.9: This figure shows 9 snapshots of a 101 frame animation. The animation is the result of applying the *Linear Regression Analysis* method to scenario 2b.

numerical results of the animation in regard to our measures are summarized in the following table:

Method	SEC	UNM	CEL	TEC	NPL	UNI	Init	T/F
Regression	105	6.805E+03	895	612	690,308	954	60	5.8
Linear	9,681	1.670E+10	202,046	582,018	439,723	3,577,834	0	1.0
Best	105	6.805E+03	750	612	439,723	749	0	1
Median	347	4.339E+08	24,396	6,301	662,911	33,607	30	4.3



The measured goodness-of-fit value in scenario 2b is $r^2 = 99\%$ for the linear transformation from the initial to the target drawing, and $r^2 = 99\%$ for the reverse transformation. The transformation from the initial to the target drawing can be described by an affine linear transformation with some additional noise in the form of small local offsets. The *Linear Regression* method successfully identifies the transformation with very high values for goodness-of-fit. The resulting animation is perceptually convincing, and the recorded measures are close to optimal in all categories but *node path length*.

Scenario 3

Figure 6.10 shows the result of applying the *Linear Regression Analysis* method to scenario 3. The

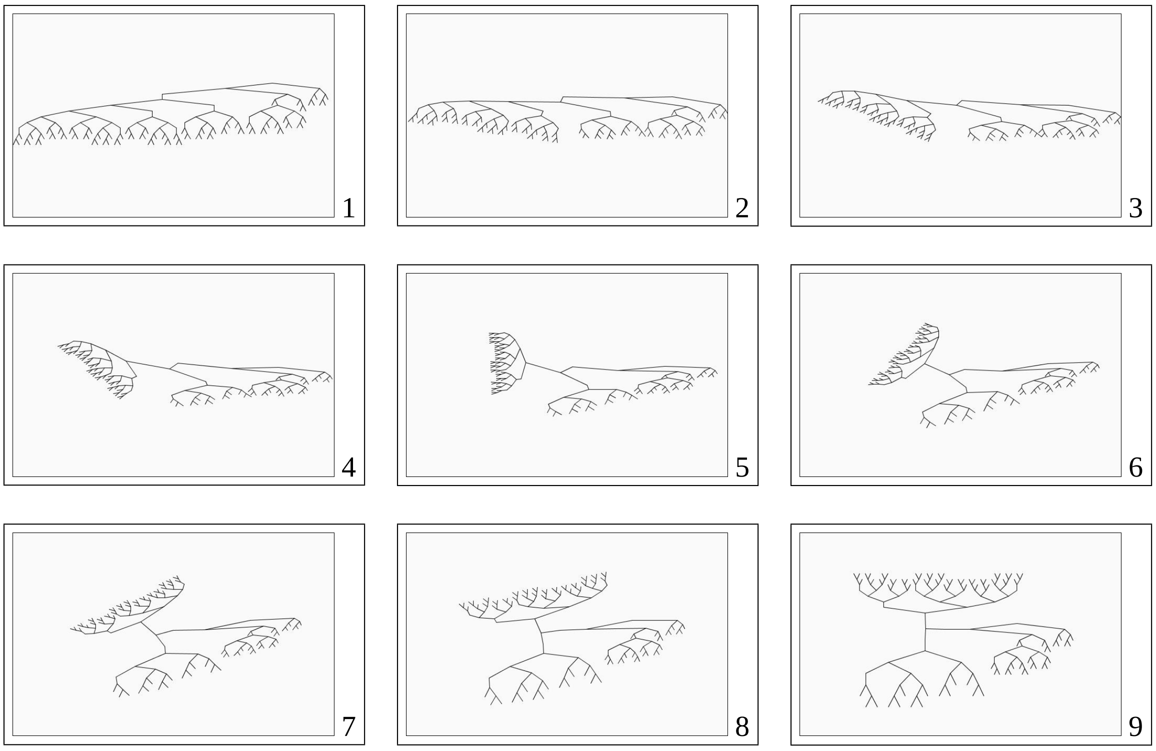
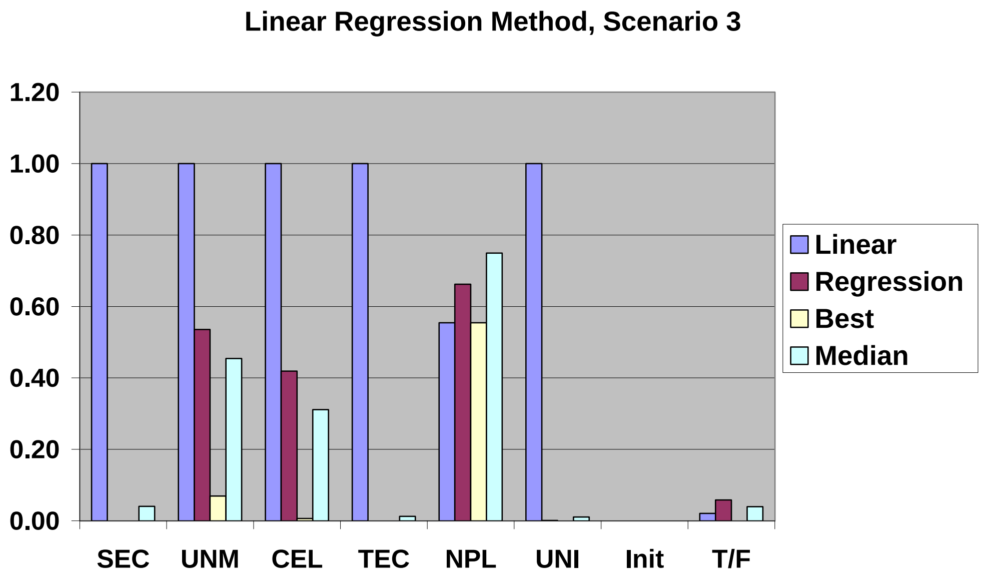


Figure 6.10: This figure shows 9 snapshots of a 101 frame animation. The animation is the result of applying the *Linear Regression Analysis* method to scenario 3.

numerical results of the animation in regard to our measures are summarized in the following table:

Method	SEC	UNM	CEL	TEC	NPL	UNI	Init	T/F
Regression	0	2.121E+08	48,123	0	113,145	19	0	1.7
Linear	40	3.962E+08	114,892	4,055	94,772	23,519	0	0.6
Best	0	2.753E+07	720	0	94,772	0	0	0
Median	2	1.799E+08	35,772	51	128,102	245	0	1.1



The measured goodness-of-fit value in scenario 3 is $r^2 = 34\%$ for the linear transformation from the initial to the target drawing, and $r^2 = 56\%$ for the reverse transformation. The low goodness-of-fit values indicates that the transformation between the initial drawing and the target drawing cannot be described well by an affine linear transformation. And indeed, the animation is clearly not optimal. The actual transformation underlying scenario 3 consists of three differently behaving sub-graphs. The *Linear Regression* method, however, computes a transformation which averages over the motions of all nodes. This leads to locally not optimal motions for most nodes. Although the recorded values are significantly better than for the *Direct Linear Interpolation* method, they are nevertheless significantly worse than for the best method.

Scenario 4

Figure 6.11 shows the result of applying the *Linear Regression Analysis* method to scenario 4. The

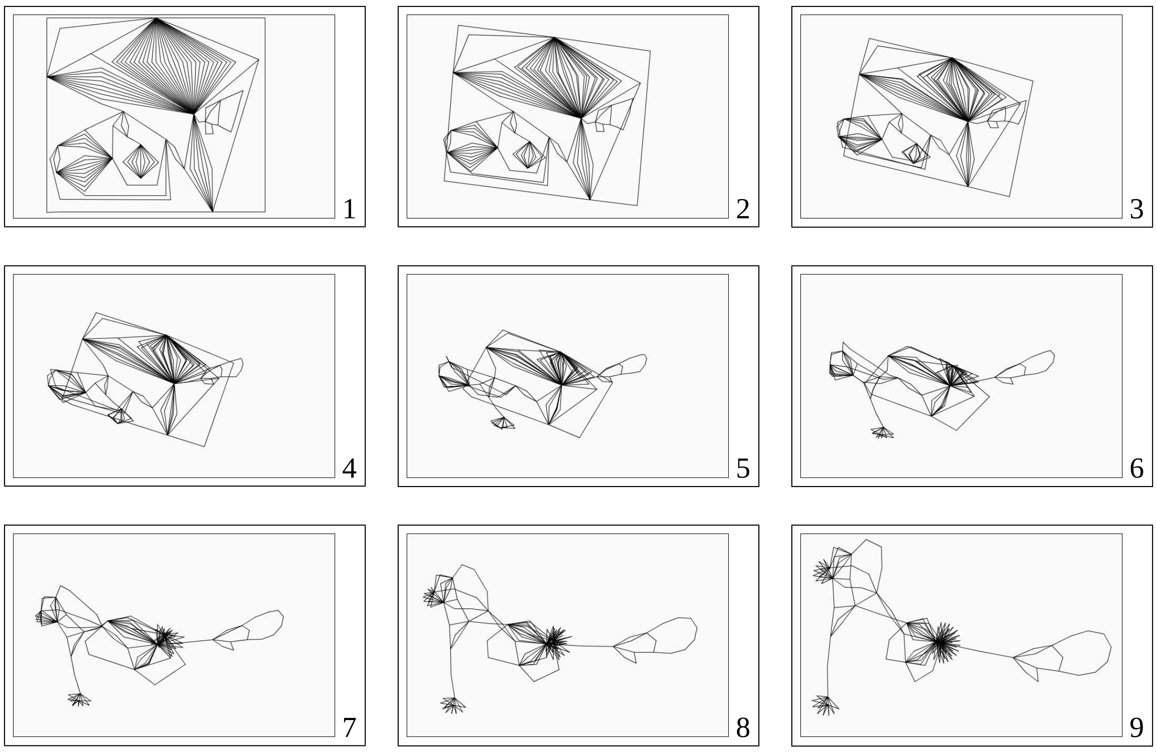
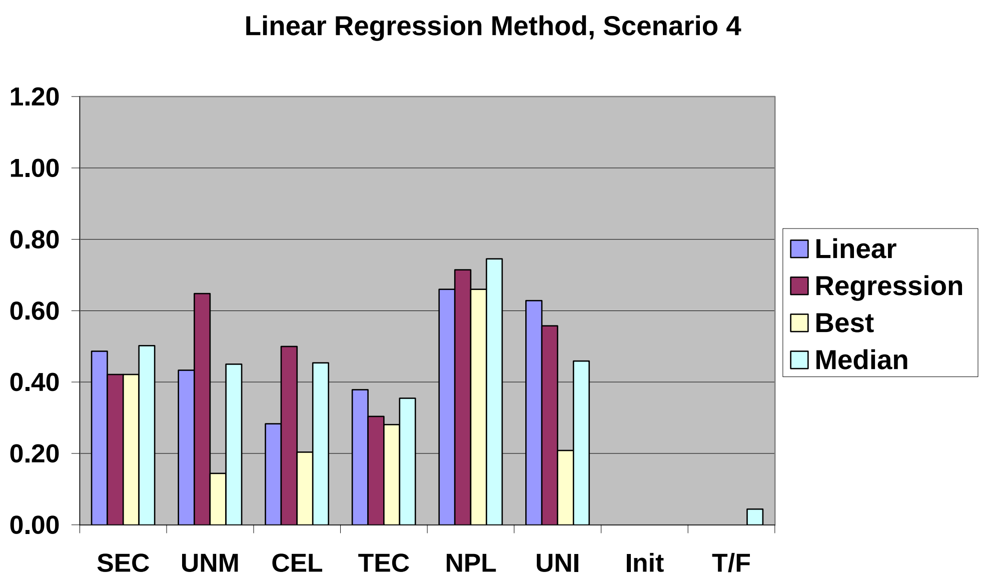


Figure 6.11: This figure shows 9 snapshots of a 101 frame animation. The animation is the result of applying the *Linear Regression Analysis* method to scenario 4.

numerical results of the animation in regard to our measures are summarized in the following table:

Method	SEC	UNM	CEL	TEC	NPL	UNI	Init	T/F
Regression	164	4.306E+08	2,341,735	1,066	130,556	150	0	0.0
Linear	190	2.877E+08	1,326,461	1,328	120,561	169	0	0.0
Best	164	9.562E+07	954,237	985	120,561	56	0	0
Median	196	2.991E+08	2,125,756	1,244	136,133	124	0	0.5



The measured goodness-of-fit value in 4 scenario is $r^2 = 49\%$ for the linear transformation from the initial to the target drawing, and $r^2 = 41\%$ for the reverse transformation. The remarks for the previous scenarios are consistent with the results for scenario 4.

Scenario 5

Figure 6.12 shows the result of applying the *Linear Regression Analysis* method to scenario 5. The

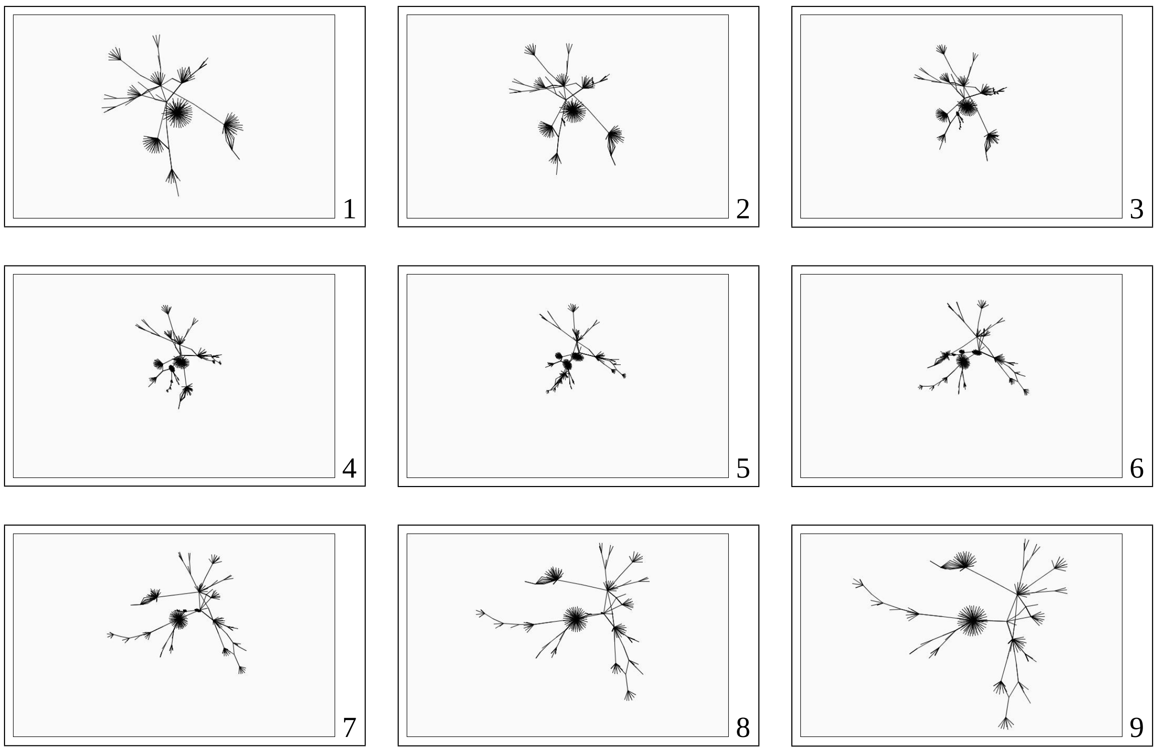
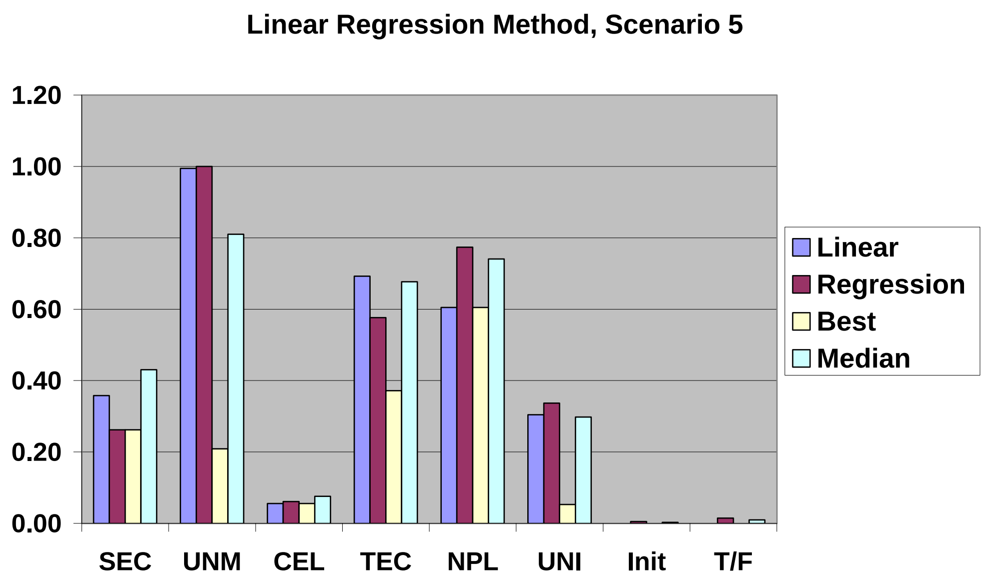


Figure 6.12: This figure shows 9 snapshots of a 101 frame animation. The animation is the result of applying the *Linear Regression Analysis* method to scenario 5.

numerical results of the animation in regard to our measures are summarized in the following table:

Method	SEC	UNM	CEL	TEC	NPL	UNI	Init	T/F
Regression	308	1.479E+10	2,626,936	13,095	763,763	307	110	1.7
Linear	420	1.470E+10	2,399,401	15,725	597,271	277	0	0.0
Best	308	3.088E+09	2,399,401	8,438	597,271	48	0	0
Median	505	1.198E+10	3,262,048	15,373	731,150	272	55	1.1



The measured goodness-of-fit value in scenario 5 is $r^2 = 21\%$ for the linear transformation from the initial to the target drawing, and $r^2 = 21\%$ for the reverse transformation. The results for scenario 5 are consistent with the remarks for the previous scenarios.

Scenario 6

Figure 6.13 shows the result of applying the *Linear Regression Analysis* method to scenario 6. The

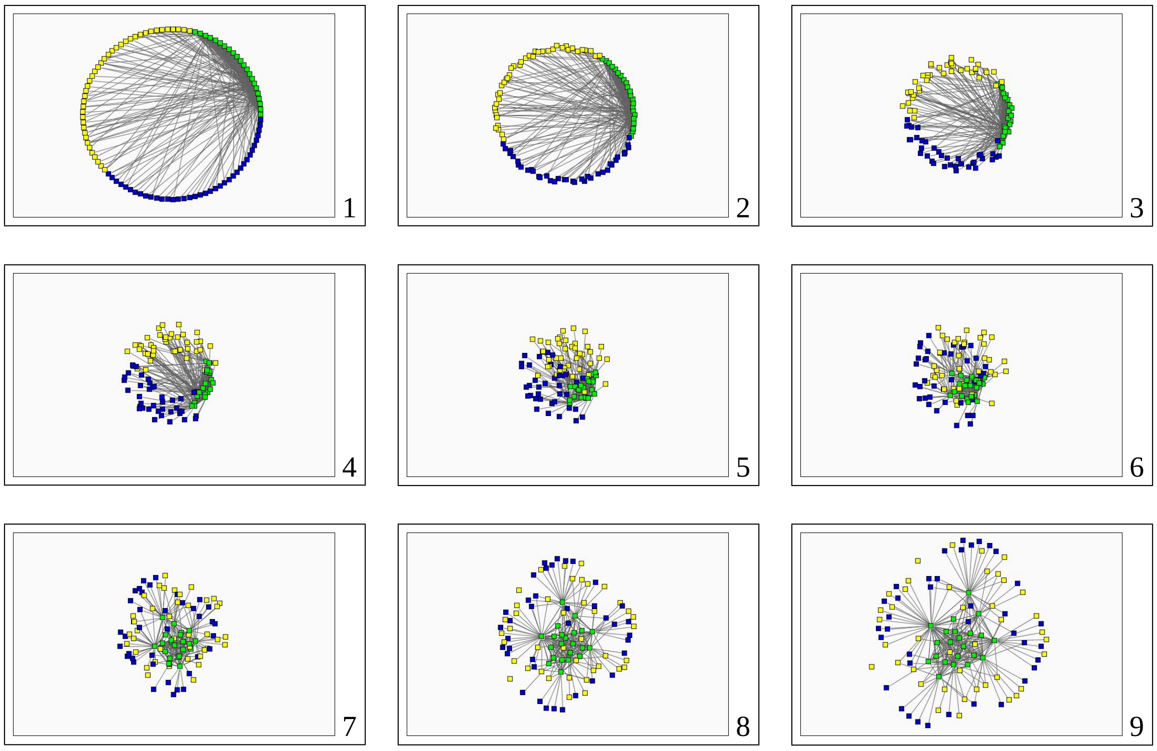
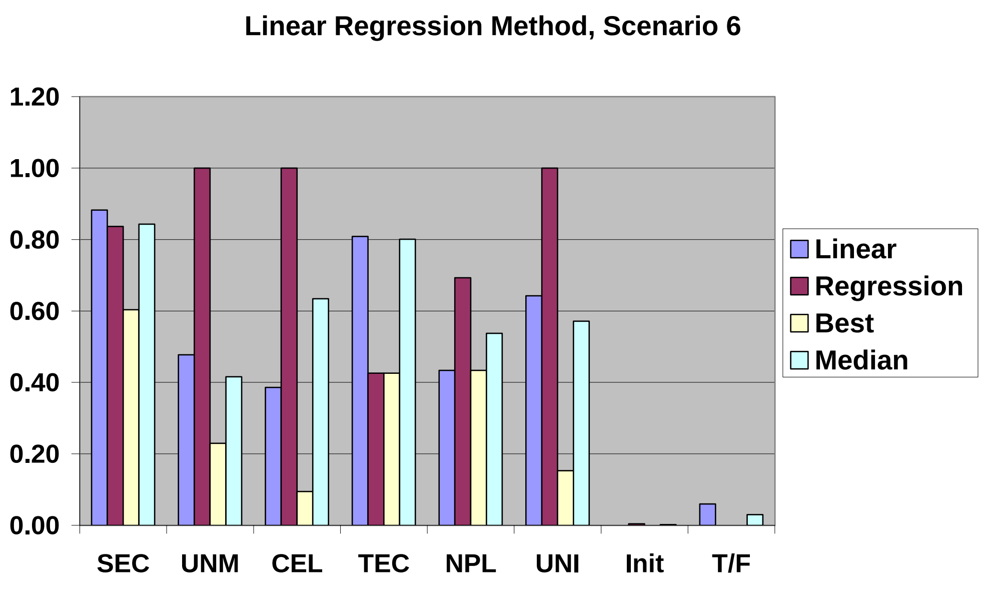


Figure 6.13: This figure shows 9 snapshots of a 101 frame animation. The animation is the result of applying the *Linear Regression Analysis* method to scenario 6.

numerical results of the animation in regard to our measures are summarized in the following table:

Method	SEC	UNM	CEL	TEC	NPL	UNI	Init	T/F
Regression	6,725	4.998E+08	10,812,041	8,301	118,195	288	50	0.0
Linear	7,095	2.385E+08	4,174,533	15,762	73,957	185	0	0.5
Best	4,853	1.147E+08	1,020,039	8,301	73,957	44	0	0
Median	6,777	2.077E+08	6,856,478	15,610	91,672	165	25	0.2



The measured goodness-of-fit value in scenario 6 is $r^2 = 0.5\%$ for the linear transformation from the initial to the target drawing, and $r^2 = 0.5\%$ for the reverse transformation. The low r^2 indicate that there is no affine linear relationship between the two drawings in this scenario.

Scenario 7

Figure 6.14 shows the result of applying the *Linear Regression Analysis* method to scenario 7. The

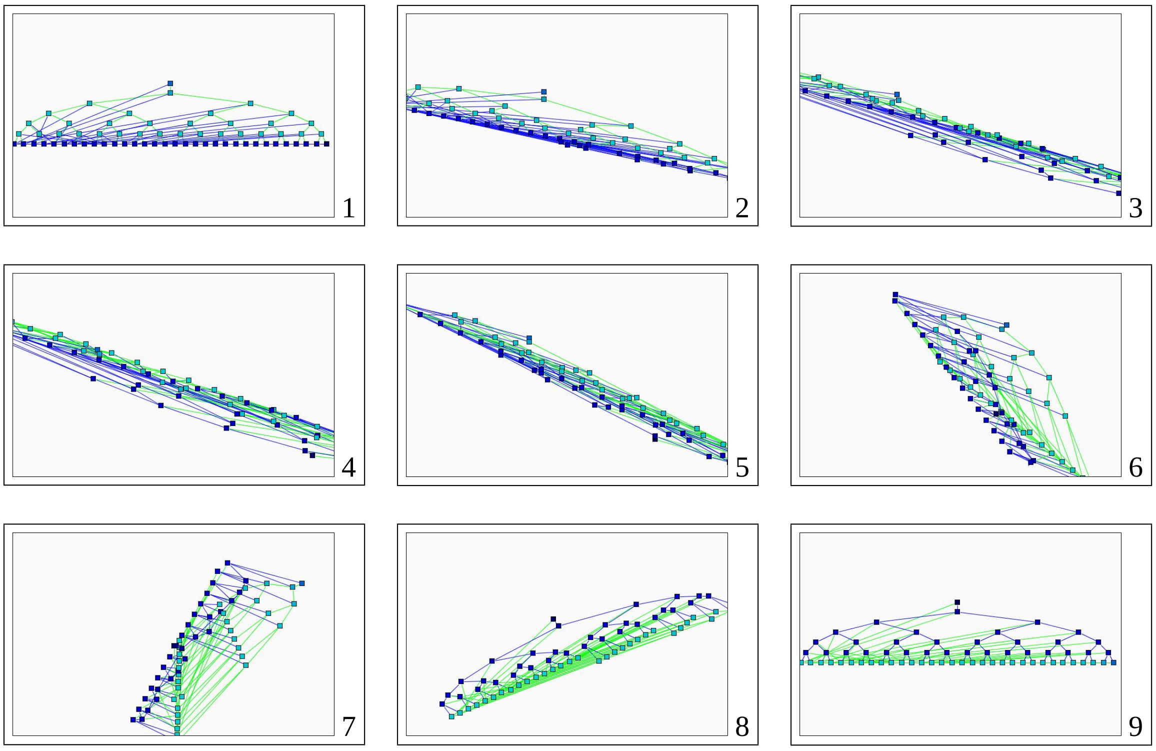
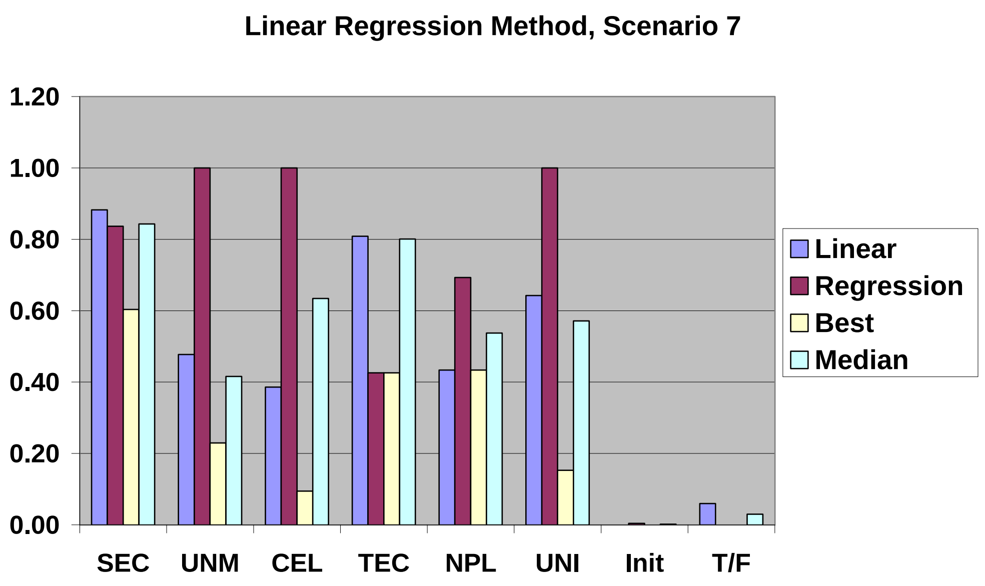


Figure 6.14: This figure shows 9 snapshots of a 101 frame animation. The animation is the result of applying the *Linear Regression Analysis* method to scenario 7.

numerical results of the animation in regard to our measures are summarized in the following table:

Method	SEC	UNM	CEL	TEC	NPL	UNI	Init	T/F
Regression	454	4.496E+07	678,300	3,718	59,502	198	50	0.0
Linear	1,011	4.858E+07	90,948	4,975	25,170	2,692	0	0.0
Best	454	1.841E+07	71,857	2,905	25,170	77	0	0
Median	696	4.262E+07	816,052	4,631	45,283	198	25	1.1



The measured goodness-of-fit value in scenario 7 is $r^2 = 29\%$ for the linear transformation from the initial to the target drawing, and $r^2 = 29\%$ for the reverse transformation. The results for scenario 7 are consistent with the remarks for the previous scenarios.

Scenario 8

Figure 6.15 shows the result of applying the *Linear Regression Analysis* method to scenario 8. The

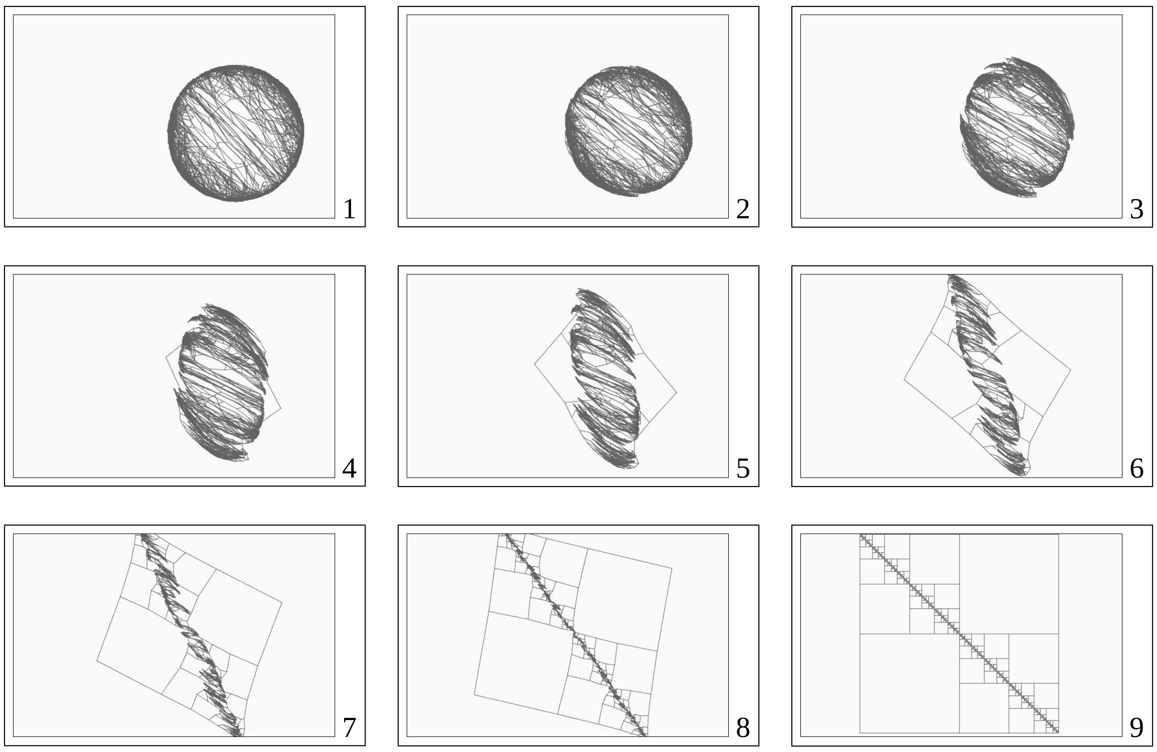
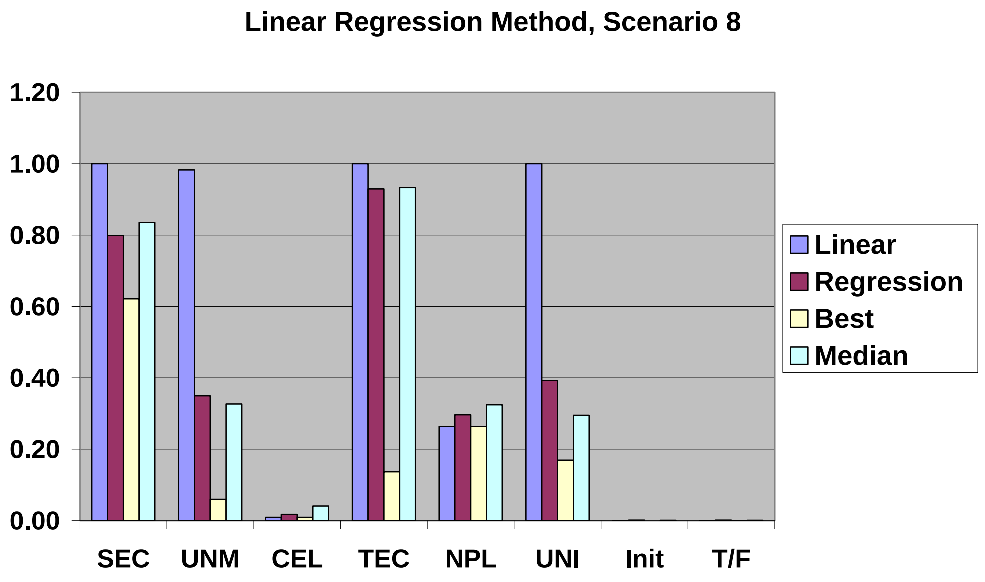


Figure 6.15: This figure shows 9 snapshots of a 101 frame animation. The animation is the result of applying the *Linear Regression Analysis* method to scenario 8.

numerical results of the animation in regard to our measures are summarized in the following table:

Method	SEC	UNM	CEL	TEC	NPL	UNI	Init	T/F
Regression	36,894	1.007E+09	7,932	1,962,955	326,451	2,915,604	110	20.1
Linear	46,185	2.829E+09	4,138	2,112,739	290,510	7,427,591	50	7.2
Best	28,706	1.718E+08	4,138	289,169	290,510	1,256,611	0	4
Median	38,574	9.407E+08	18,486	1,971,003	357,359	2,192,830	80	13.7



The measured goodness-of-fit value in scenario 8 is $r^2 = 87\%$ for the linear transformation from the initial to the target drawing, and $r^2 = 46\%$ for the reverse transformation. We note a particularly high difference between the goodness-of-fit values between the two computed affine linear transformations. Many nodes in the target drawing lie on, or close to, a straight line. In this respect, scenario 8 is quite similar to example in figure 6.4 on page 211. The large difference between the goodness-of-fit values is therefore not surprising.

6.1.5 Summary

Minimizing the sum of the Euclidean squares is known to be statistically biased and to overemphasize outlying samples. On the other hand, other distance functions are much harder to minimize and generally require numerical methods [54, 91]. To increase statistic stability different approaches are possible. The median or center of the drawings could be added to the node set with a weight factor. Another promising approach would be to compute different transformations using random subsets of the nodes and using the best one. Despite this shortcoming, the *Linear Regression Analysis* method produces good results in almost all scenarios.

The *Linear Regression* method is designed to identify and animate changes between drawings of graphs which can be described by an affine linear transformation. These in particular include transformations with a significant rotational component. The *Linear Regression Analysis* method achieves very good results in these scenarios. For general scenarios, the *Linear Regression Analysis* method produces results which are at least as good as the *Direct Linear Interpolation* method for almost all scenarios and measures. The *Linear Regression Analysis* method averages the overall motion of all nodes of the graph. As a consequence it achieves rather poor results if sub-graphs show different kinds of motions, as we have seen in scenario 3 (Figure 6.10 on page 221). In these cases it can produce animations which achieve worse values than the *Direct Linear Interpolation* method.

The efficiently computable goodness-of-fit value r^2 can be taken into account when deciding whether to use the *Linear Regression Analysis* method for a given scenario. Although we were not able to identify a clear threshold, in our experience r^2 values of more than 70% generally indicate a good result for the *Linear Regression Analysis* method.

6.2 Motion Cluster Analysis Method

Although the *Linear Regression Analysis* method works well when the motion of the nodes is more or less uniform, it performs poorly when several subgraphs have very different underlying motions, as is generally the case when only part of a graph layout is updated. The method attempts to compute an average over all motions of individual nodes, and as a result can decide upon a motion that is not well suited to all parts of the graph. Figure 6.16 shows an example where computing the best overall affine linear transformation produces a poor result.

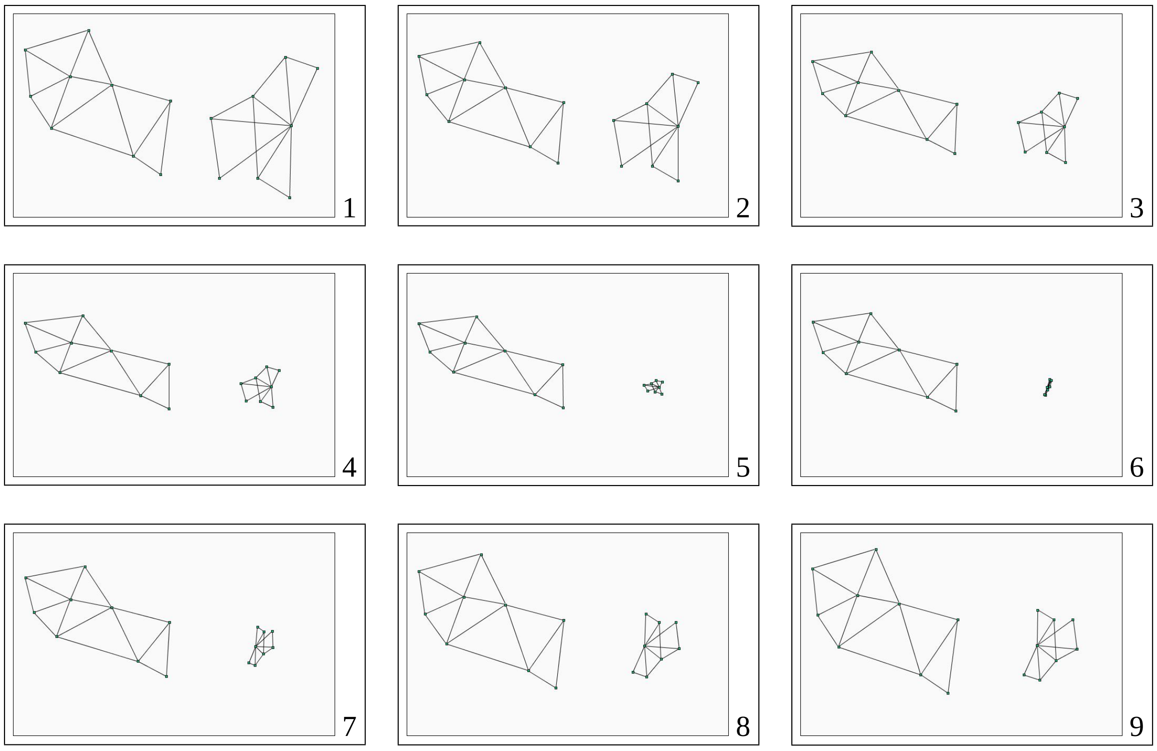


Figure 6.16: Example of a bad animation, starting at Frame 1 and ending at Frame 9. Although only the right-hand component of the graph should move, the calculation and application of an average movement causes distortion in the left-hand component.

In this section, we use linear regression analysis as the basis of several clustering heuristics which seek to identify subgraphs which share a similar, structured motion. Applying different transformations to the subgraphs identified allows for the partial update of graph layouts. Figure 6.17 shows the result of applying the method introduced in Section 6.2.2 to the situation of Figure 6.16.

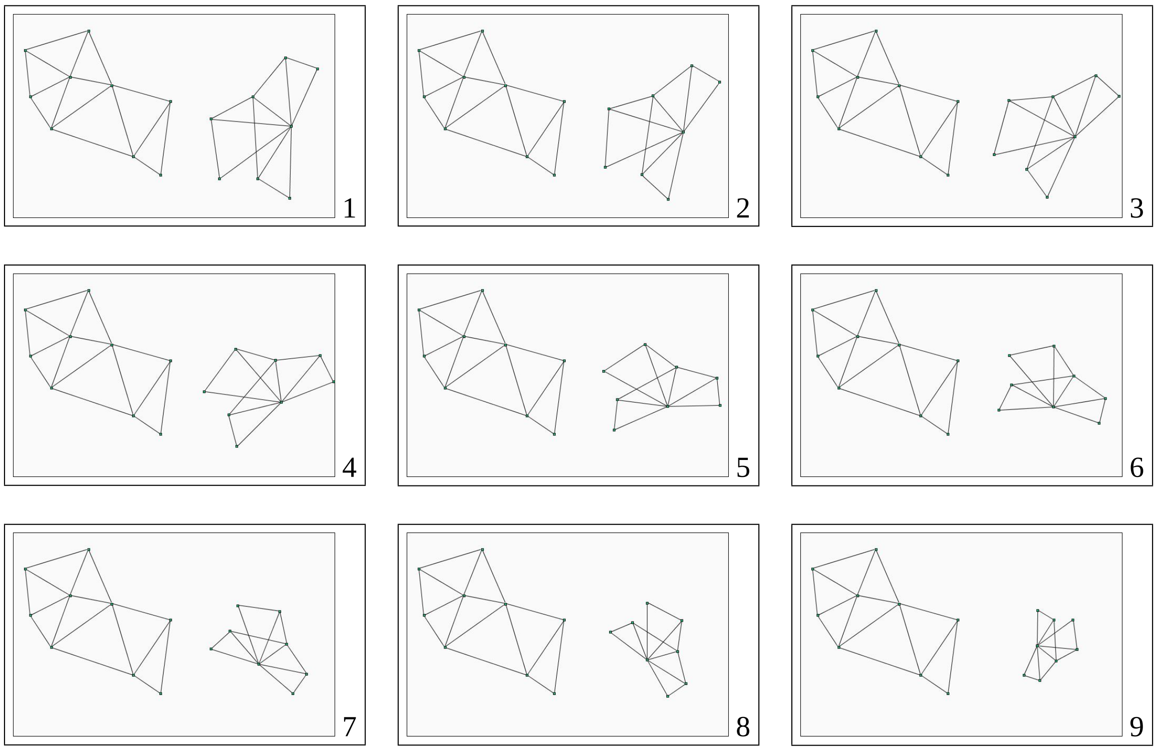


Figure 6.17: Example of a good animation. Different motions are computed for each of the components of the graph. The result is an animation where each subgraph moves in an individual and much more intuitive way.

6.2.1 Determining Candidate Transformations

For each node v there is an infinite number of affine linear transformations that can take it from its initial position to its target position. If the transformations applied to individual nodes were chosen arbitrarily and independently, the resulting animation would appear chaotic. In order to effect a smooth animation, it stands to reason that nodes should be grouped together under common transformations whenever possible.

Using this principle, we can restrict the candidate transformations for v to those that v might share with other nodes. In general, three nodes (associated with three non-collinear initial locations and three non-collinear target locations) are required to uniquely determine an affine linear transformation in the two-dimensional plane. Naturally, if four or more nodes were to determine a common affine transformation, this transformation would also be determined by at least one of its subsets of cardinality three. This suggests that the candidate transformations for v could be limited to those generated by triples of nodes that include v. However, the total number of transformations generated would be cubic in the number of nodes. For the purposes of real-time animation, this candidate set would still be far too large to be investigated explicitly.

To avoid paying the cubic cost of generating all candidate transformations of node triples, practical heuristics are needed. In the *Cluster Analysis* animation method, we use clustering techniques to identify subsets of nodes which share similar transformations.

6.2.2 k-Means

The first clustering method we use is based on the well-known k-means hill-climbing heuristic [69]. The general k-means heuristic for point sets begins with an arbitrary partition $\mathcal{P} = \{P_1, P_2, \ldots, P_k\}$ of the data set into k groups. It then attempts an iterative improvement of the partition, as follows:

- a representative point is computed for each group P_i;
- each element of the data set is assigned to the representative that best suits it, generating a new partition $\mathcal{P}' = \{P'_1, P'_2, \ldots, P'_k\}$.

The process is repeated until an iteration yields no improvement, according to some measure of goodness of a partition.

It is easy to see that this procedure must eventually converge. The great popularity of the k-means method is due to its simplicity and speed. Typically, the number of iterations required is constant, leading to an observed linear-time complexity.

Our adaptation of k-means computes an affine transformation of each group of nodes as the representative of the group, using the linear regression techniques outlined in section 6.1.1. In the second step of each iteration, the redistribution of nodes is accomplished by assigning each node to the representative transformation that brings the node closest to its final destination. Improvement can be evaluated according to such measures as the sum of these distances to final destinations, or of squares of these distances. Iteration continues until no further improvement is made.

6.2.3 The Method

We can therefore propose the following animation method using k-means clustering:

1. Apply k-means to determine clusters of uniformly moving nodes:

    - Randomly distribute nodes over k clusters

- For each cluster compute a transformation by applying linear regression analysis to the nodes of the cluster
- Until convergence
  - Redistribute each node to whatever cluster describes its transformation best
  - Recompute the transformations for each cluster according to its new members

2. Assign to each node the transformation according to its clusters. The final offset between the result of the transformation and the actual position of the node in the target drawing is added to the matrix as a translation

3. Perform polar decomposition on all transformations.

4. Interpolate the path of each node according to its associated transformation.

   Given a graph $G = (V, E)$ with $n = |V|$, an affine linear transformations $T_1 = (R_1, S_1)$, $T_2 = (R_2, S_2), \ldots, T_n = (R_n, S_n)$ describing the motion of each node from the initial to the target layout, each consisting of an rotational angle R_i and a non rotational matrix S_i, the position of each node $v \in V$ with initial position $P_0(v)$ at time t is given by:

$$P(v,t) = p_0(v)(1-t)M(tR_v)(I + tS_v)$$

   with M being a function transforming an angle into its corresponding rotational matrix and I being the identity matrix.

k The k-means clustering method requires that the number of clusters k is specified at the beginning. In all our examples and experiments k was set to 10, even though the number of different transformations involved was significantly less. This choice was motivated by two observations. First, as the short-term memory of most users is believed to be able to accommodate roughly 7 items [76], we conclude that computing more than 10 very distinct transformations in one animation would not likely lead to a better preservation of the user's mental map. Second, sub-clusters derived from a well-associated larger cluster will be associated with very similar transformations, provided that each sub-cluster contains at least three linearly-independent nodes. This indicates that the method will likely tolerate a choice of k somewhat larger than the minimum.

6.2.4 Analysis

1. Init

   Given a graph $G = (U, V)$, $n = |V|$ and k clusters, the following steps have to be performed:

   - The nodes are randomly distributed over k clusters. This takes $O(n)$ time.
   - For each cluster compute a transformation by applying linear regression analysis to the nodes of the cluster. Computing the linear regression for a cluster is linear to the number of elements in the cluster. As each node belongs to exactly one cluster, this amounts to a total computation time of $O(n)$.
   - The clustering is iteratively improved using the following loop until it converges.
     – Each node is redistributed to whatever cluster describes its transformation best. This takes $O(n)$ time.
     – The transformations for each cluster have to be recomputed according to its new members. As discussed before, this takes $O(n)$ time.
   - After the clustering, each node is assigned a transformation according to the cluster to which it belongs. A final translation is added to compensate for the difference between the result of the affine linear transformation and the actual target position of the node. This takes $O(n)$ time.
   - All transformations are decomposed in a total of $O(n)$ time using the polar decomposition method.

   Assuming that k-means converges in a constant number of iterations we have $O(n)$ running time for *Init*.

2. T/F

   The position of every node can be updated in constant time, resulting in a time complexity of $O(n)$ to compute a frame.

3. SEC, TEC, UNI

   Within each cluster the same results as for the *Linear Regression* method apply. That is, if the target drawing of a cluster can be well described by an affine linear transformation of the initial drawing of that cluster, then *static edge crossings*, *temporary edge crossings*, and *unnecessary node intersections* have close to optimal values. However, no mechanisms exist in the *Cluster Analysis* animation method which help to optimize *static edge crossings*, *temporary edge crossings*, and *unnecessary node intersections* between clusters. Scenarios can be constructed which result in unnecessarily bad values for these measures. Figure 6.18 shows such an example. As with the scenario in figure 6.4 on page 211, the right subgraph

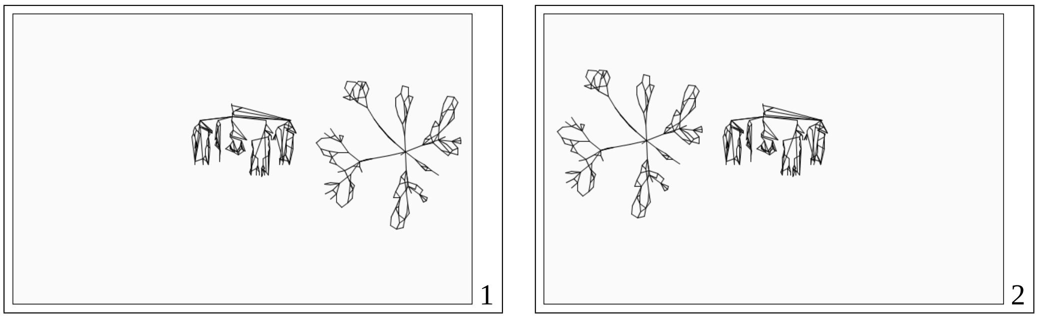


Figure 6.18: Example for a scenario where the *Cluster Analysis* method is able to correctly identify the different motion clusters, but nevertheless bad values for most measures.

moves to the left, while the left subgraph stays at its original position. The corresponding animation is shown in figure 6.19. The *Cluster Analysis* method is able to identify the correct motion clusters and animates them with perfect values for most measures within each cluster. However, as the *Cluster Analysis* method does not provide any mechanisms for considering inter-cluster relationships, the cluster move through each other which results in bad values for most measures.

4. CEL, UNM

   Again, the same results as for the *Linear Regression* method apply within each cluster. As for *static edge crossings*, *temporary edge crossings*, and *unnecessary node intersections* the *Cluster Analysis* method does not provide any mechanisms to support good overall values for *constant edge length* and *uniform node movement*. On the contrary, the *Cluster Analysis* method deliberately accepts bad inter-cluster values for *uniform node movement* to improve the results for *uniform node movement* within each cluster. The idea behind this strategy is that by having high *uniform node movement* within each cluster, the human brain automatically interprets a whole cluster as one moving entity. The then relative small amount of interferences between clusters as single objects are easily understood by the brain. The *constant edge length* measure is equivalent to applying the *uniform node movement* measure to a

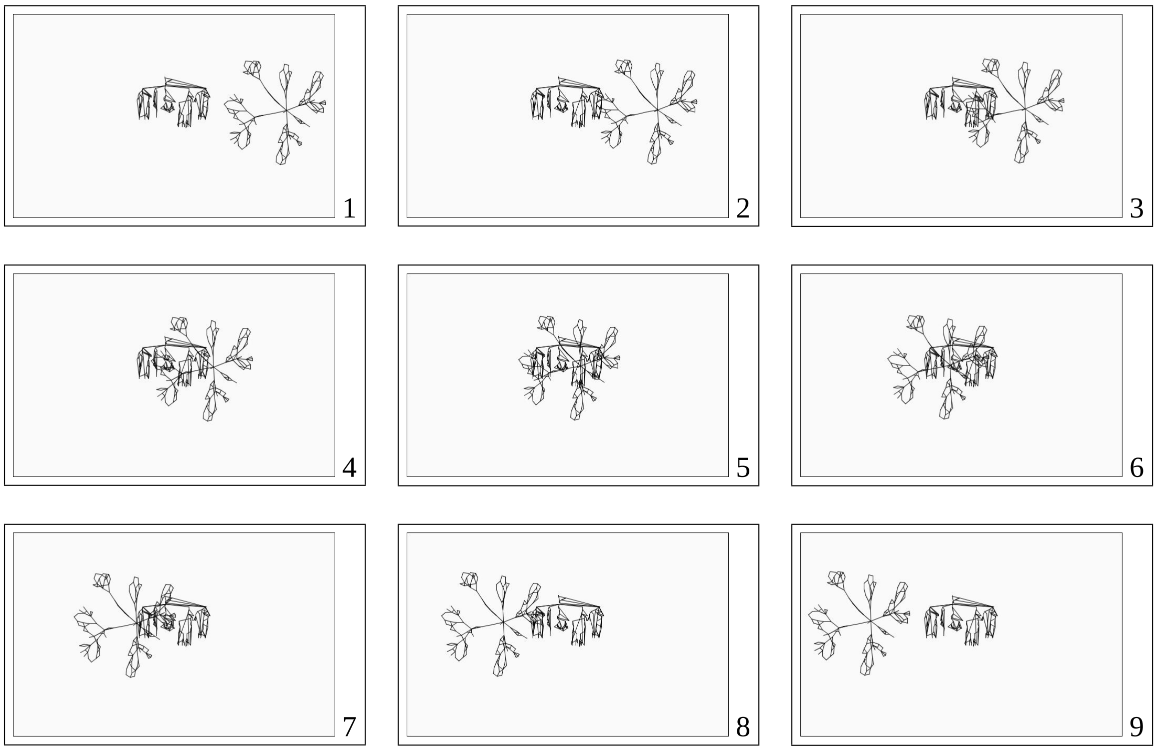


Figure 6.19: This figure shows 9 snapshots of a 101 frame animation of the scenario displayed in figure 6.18. The animation is the result of applying the *Cluster Analysis* animation method. The *Cluster Analysis* method is able to identify the correct motion clusters and animates them with perfect values for most measures within each cluster. However, as the *Cluster Analysis* method does not provide any mechanisms for considering inter-cluster relationships, the cluster move through each other which results in bad values for most measures.

subset of the node pairs of the graph. As the clustering process does not consider the edges of the graph, the values for *constant edge length* depend mainly on how similar the identified motion clusters are to the edge connectivity clusters of the graph. If the motion clusters are identical to the edge connectivity clusters, then most of the edges of the graph lie within the clusters. This results in good overall values for *constant edge length*. However, if there are many edges between motion clusters, then the values for *constant edge length* can become very bad. It is easy to construct scenarios where this happens.

5. Smoothness

   The path of each node is given by a linear transformation and is therefore smooth. In multiple key frame scenarios a non smooth direction change is expected at every key frame.

6.2.5 Empirical Evaluation

Scenario 1

Figure 6.20 shows the result of applying the *Motion Cluster Analysis* method to scenario 1. The

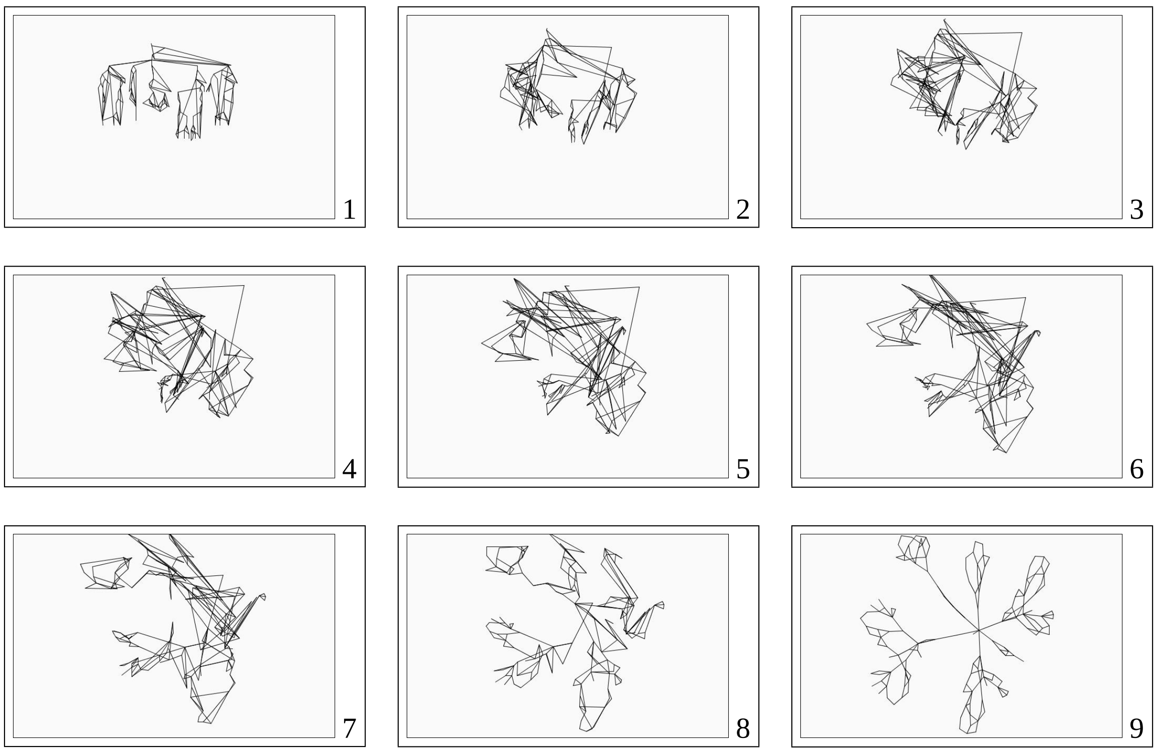
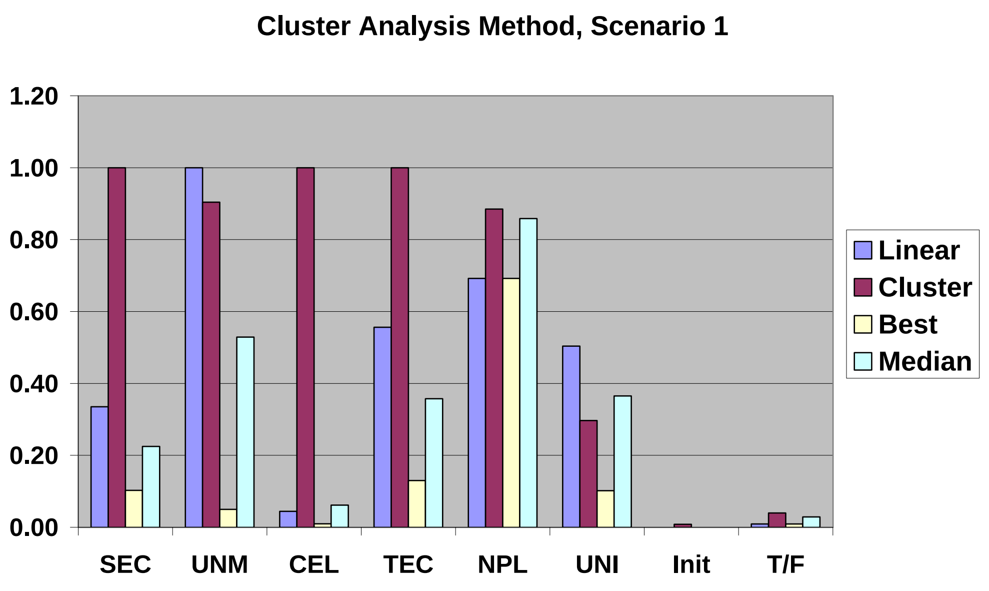


Figure 6.20: This figure shows 9 snapshots of a 101 frame animation. The animation is the result of applying the *Motion Cluster Analysis* method to scenario 1.

numerical results of the animation in regard to our measures are summarized in the following table:

Method	SEC	UNM	CEL	TEC	NPL	UNI	Init	T/F
Cluster	388	4.538E+09	41,883,085	3,765	445,023	76	270	2.2
Linear	130	5.020E+09	1,843,276	2,095	347,983	129	0	0.5
Best	40	2.484E+08	402,170	490	347,983	26	0	0
Median	87	2.656E+09	2,577,963	1,346	431,802	94	0	1.6



Although the *Cluster Analysis* method is able to identify several motion clusters with high r^2 values, the animation itself looks poor. The recorded measures show bad values for *static edge crossings*, *constant edge length*, and *temporary edge crossings*. The identified motion clusters clearly do not correspond to the intuitive transformation of the graph. However, the results of the *Cluster Analysis* method vary significantly depending on the initial random distribution of clusters. In other runs the *Cluster Analysis* method was able to produce animations for scenario 1 which are comparable in quality to the *Linear Regression* method. Nevertheless, for scenario 1 and scenario 1b, good results are not more likely than bad results when using the *Cluster Analysis* method. We thus decided to use the results of the first runs for both scenarios in our evaluation although they do not reflect the whole range of possible results.

Scenario 1b

Figure 6.21 shows the result of applying the *Motion Cluster Analysis* method to scenario 1b. The

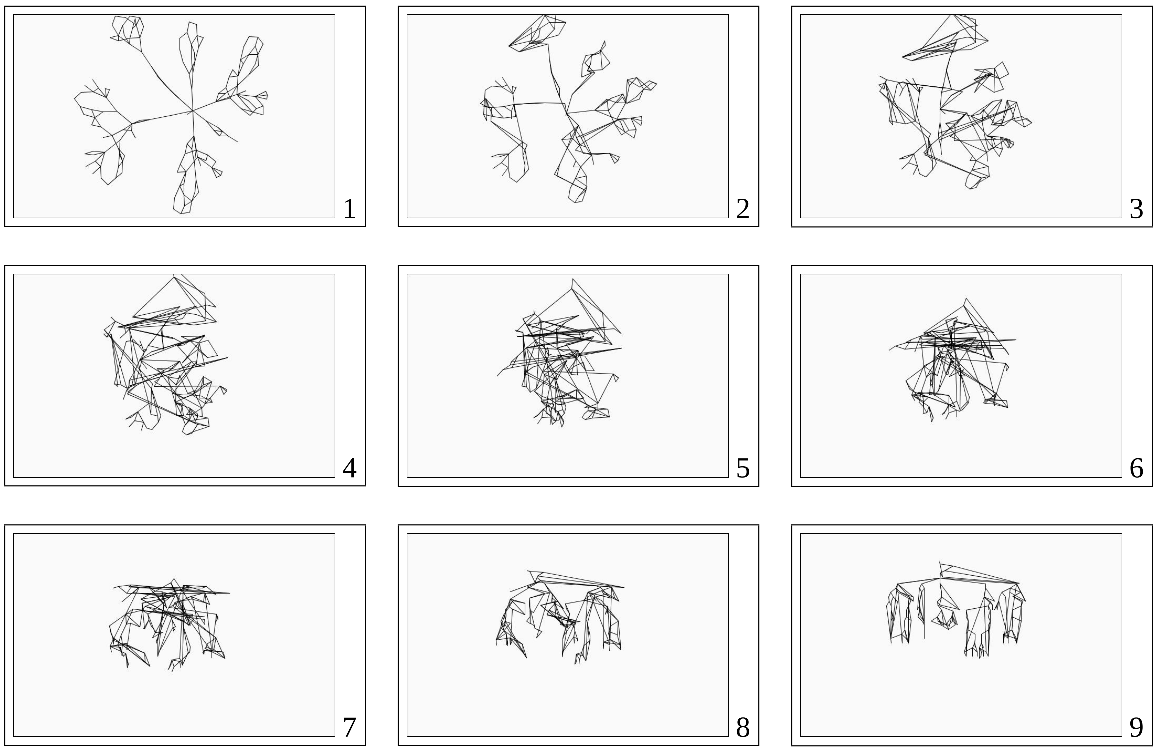
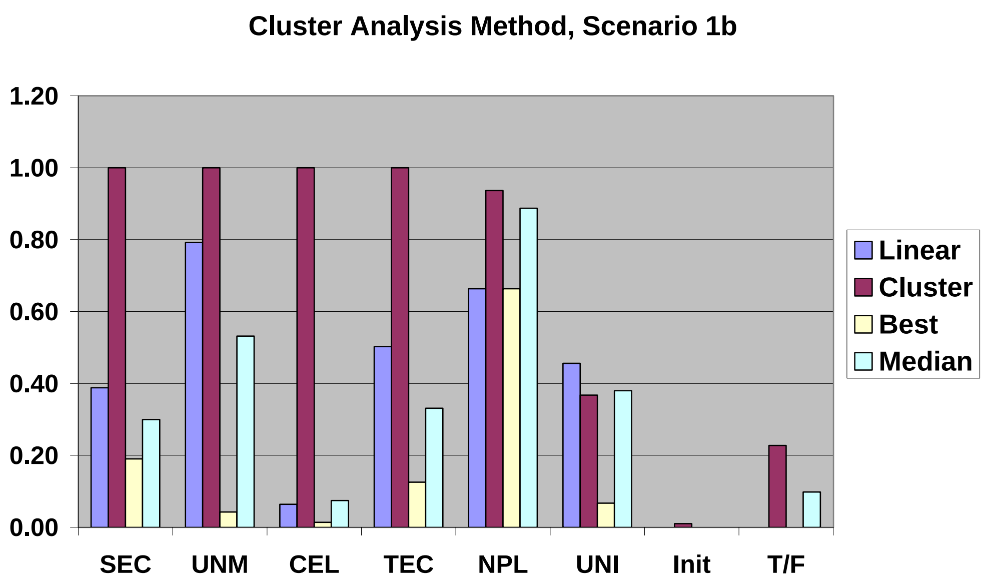


Figure 6.21: This figure shows 9 snapshots of a 101 frame animation. The animation is the result of applying the *Motion Cluster Analysis* method to scenario 1b.

numerical results of the animation in regard to our measures are summarized in the following table:

Method	SEC	UNM	CEL	TEC	NPL	UNI	Init	T/F
Cluster	335	6.340E+09	28,847,622	4,254	491,297	104	380	11.9
Linear	130	5.020E+09	1,843,276	2,138	347,983	129	0	0.0
Best	64	2.697E+08	395,880	533	347,983	19	0	0
Median	100	3.371E+09	2,146,880	1,409	465,534	108	0	5.1



The same remarks as for scenario 1 apply to scenario 1b.

Scenario 2

Figure 6.22 shows the result of applying the *Motion Cluster Analysis* method to scenario 2. The

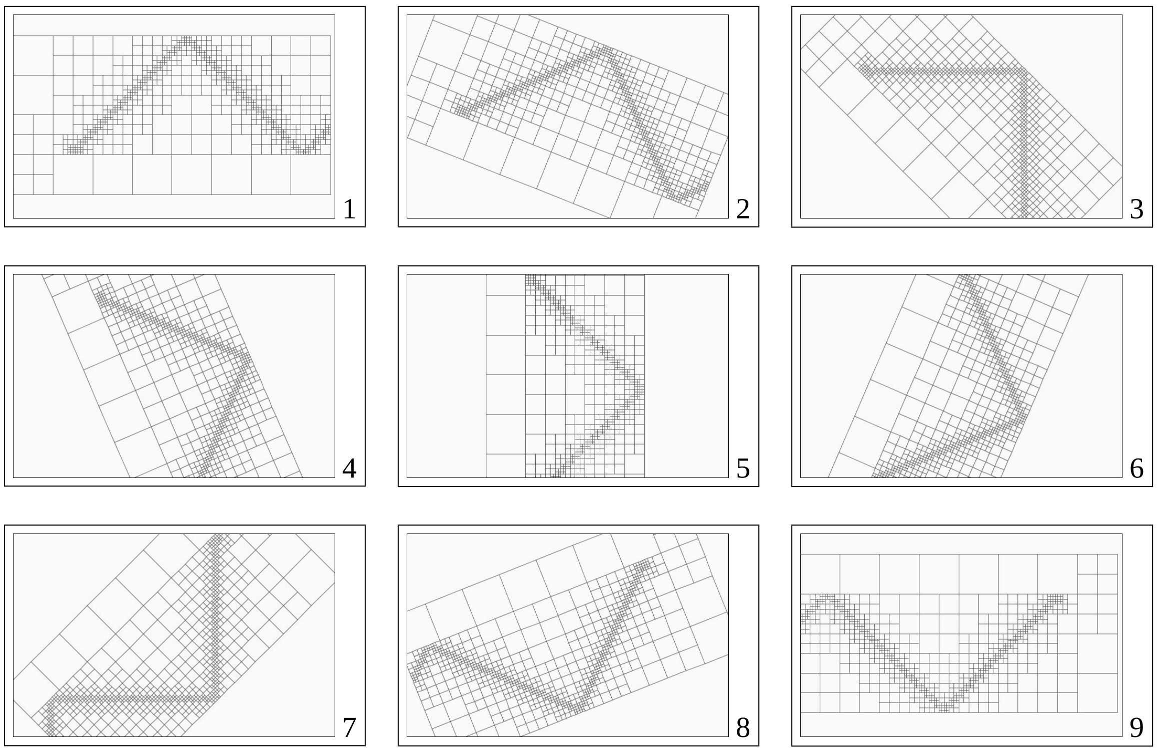
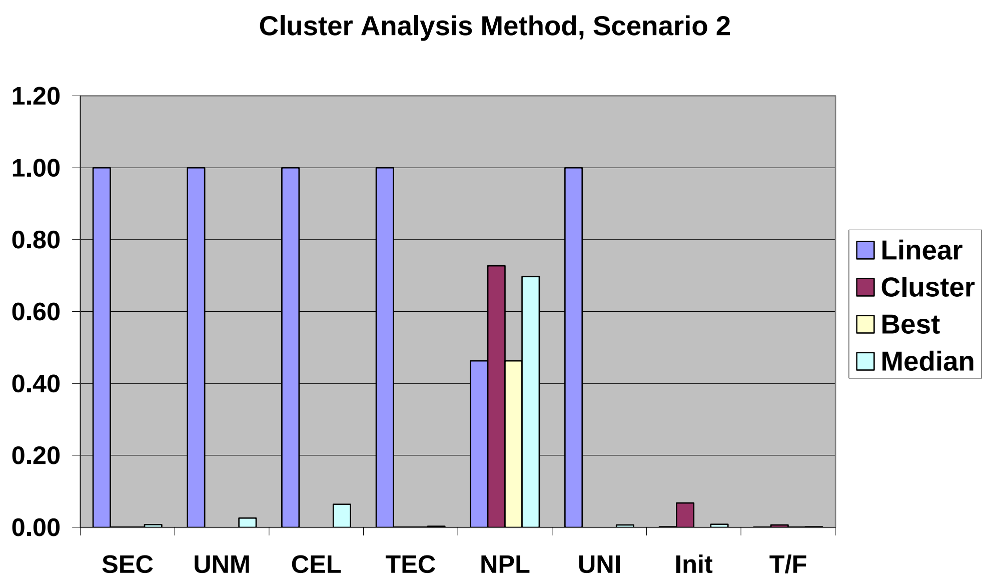


Figure 6.22: This figure shows 9 snapshots of a 101 frame animation. The animation is the result of applying the *Motion Cluster Analysis* method to scenario 2.

numerical results of the animation in regard to our measures are summarized in the following table:

Method	SEC	UNM	CEL	TEC	NPL	UNI	Init	T/F
Cluster	6	1.234E-20	0	560	691,294	0	2,200	19.0
Linear	12,891	1.675E+10	212,992	1,301,941	440,110	3,813,032	50	1.6
Best	4	1.066E-20	0	374	440,110	0	0	1
Median	92	4.259E+08	13,608	3,468	662,794	24,579	270	4.7



The *Cluster Analysis* method computes a perceptually very pleasing animation for scenario 2. The recorded measures are also very good. The *Cluster Analysis* method identifies more than one cluster in this scenario. However, as the transformations for all clusters are identical, the animation displays a uniform movement of the whole graph.

Scenario 2b

Figure 6.23 shows the result of applying the *Motion Cluster Analysis* method to scenario 2b. The

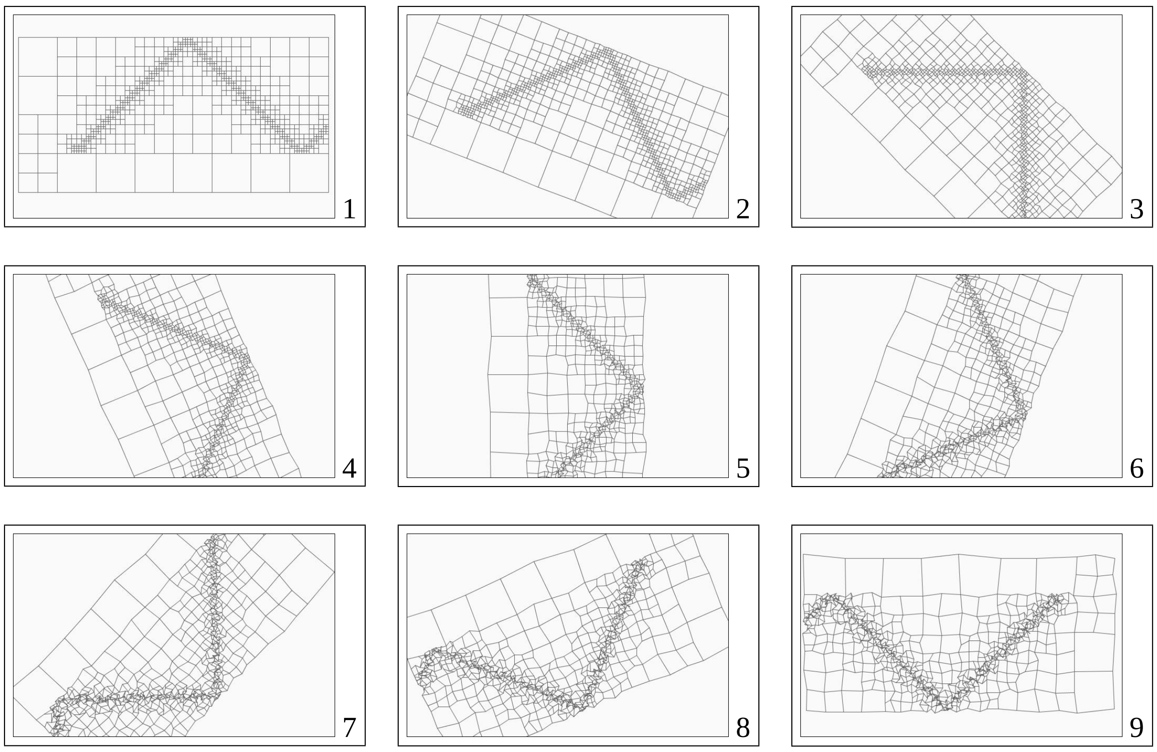
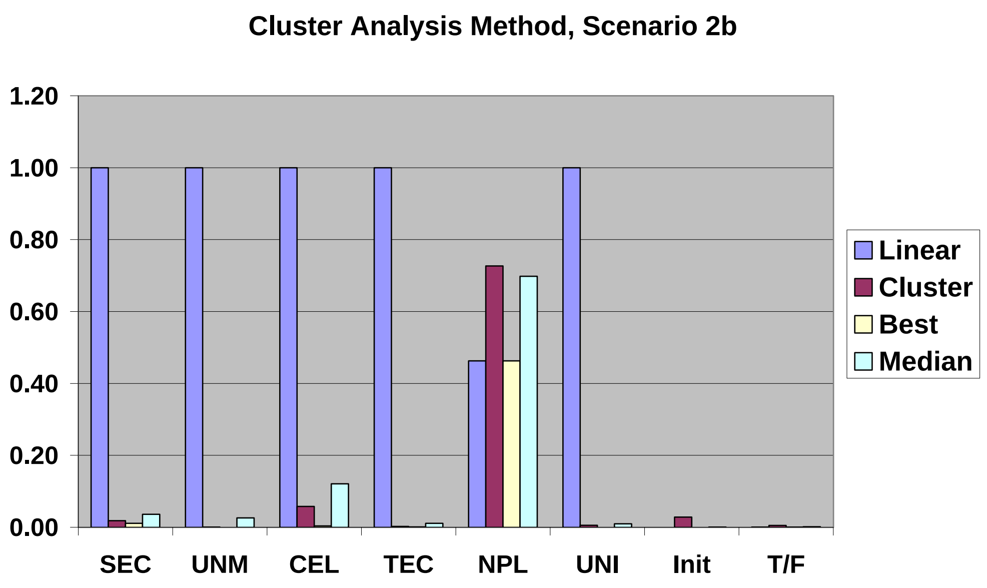


Figure 6.23: This figure shows 9 snapshots of a 101 frame animation. The animation is the result of applying the *Motion Cluster Analysis* method to scenario 2b.

numerical results of the animation in regard to our measures are summarized in the following table:

Method	SEC	UNM	CEL	TEC	NPL	UNI	Init	T/F
Cluster	176	4.184E+06	11,733	1,398	690,412	20,013	3,460	17.8
Linear	9,681	1.670E+10	202,046	582,018	439,723	3,577,834	0	1.0
Best	105	6.805E+03	750	612	439,723	749	0	1
Median	347	4.339E+08	24,396	6,301	662,911	33,607	30	4.3



The same remarks as for scenario 2 apply to scenario 2b. However, in this scenario uniform movement of the graph is not necessarily guaranteed. Due to the noise, it is for example theoretically possible that the *Cluster Analysis* method identifies clusters with opposite directions of rotation. In such a case, the graph would not move uniformly. Additional measures would be necessary to suppress such behavior. However, devising such measures is not trivial.

Scenario 3

Figure 6.24 shows the result of applying the *Motion Cluster Analysis* method to scenario 3. The

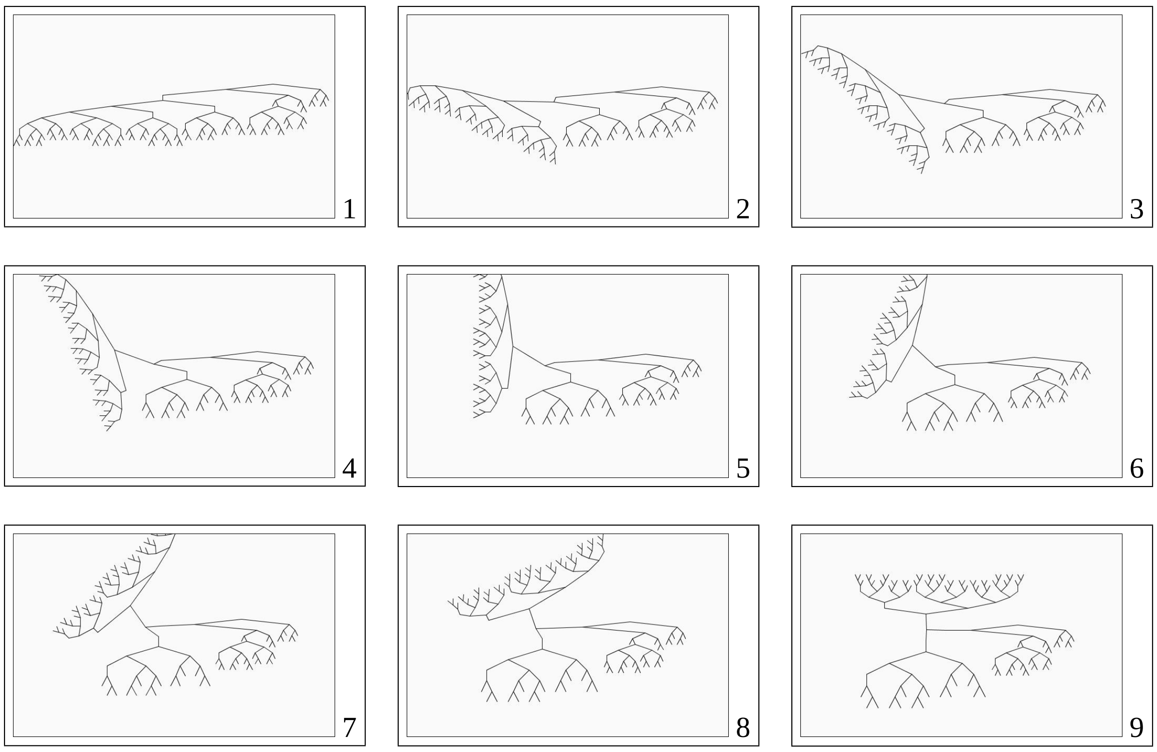
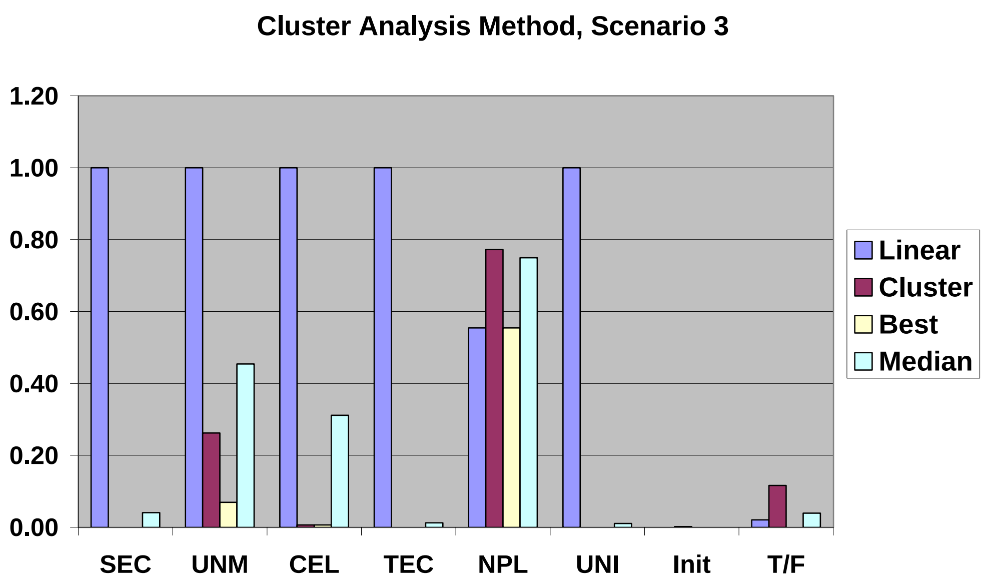


Figure 6.24: This figure shows 9 snapshots of a 101 frame animation. The animation is the result of applying the *Motion Cluster Analysis* method to scenario 3.

numerical results of the animation in regard to our measures are summarized in the following table:

Method	SEC	UNM	CEL	TEC	NPL	UNI	Init	T/F
Cluster	0	1.039E+08	720	0	132,007	0	110	3.4
Linear	40	3.962E+08	114,892	4,055	94,772	23,519	0	0.6
Best	0	2.753E+07	720	0	94,772	0	0	0
Median	2	1.799E+08	35,772	51	128,102	245	0	1.1



Scenario 3 contains three differently transformed subgraphs. All three transformations can be described by an affine linear function. The *Cluster Analysis* method is designed for exactly such cases. It successfully identifies and animates all motion clusters and achieves optimal values for *static edge crossings*, *temporary edge crossings*, and *unnecessary node intersections*.

Scenario 4

Figure 6.25 shows the result of applying the *Motion Cluster Analysis* method to scenario 4. The

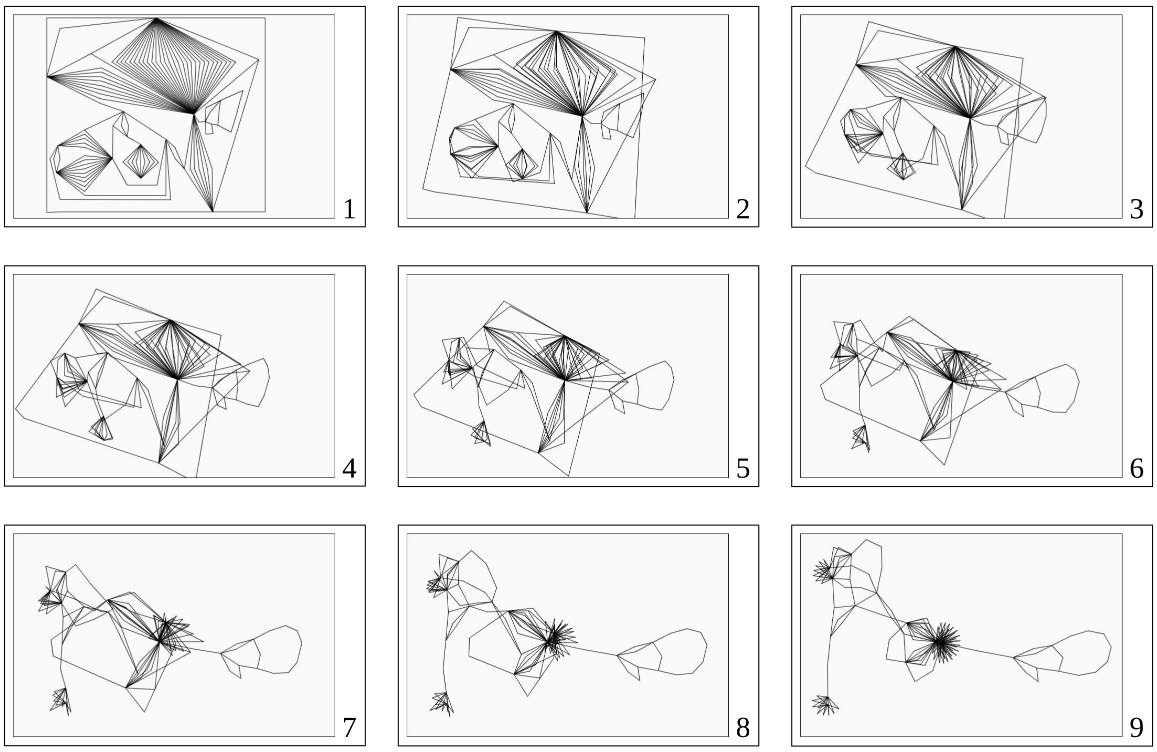
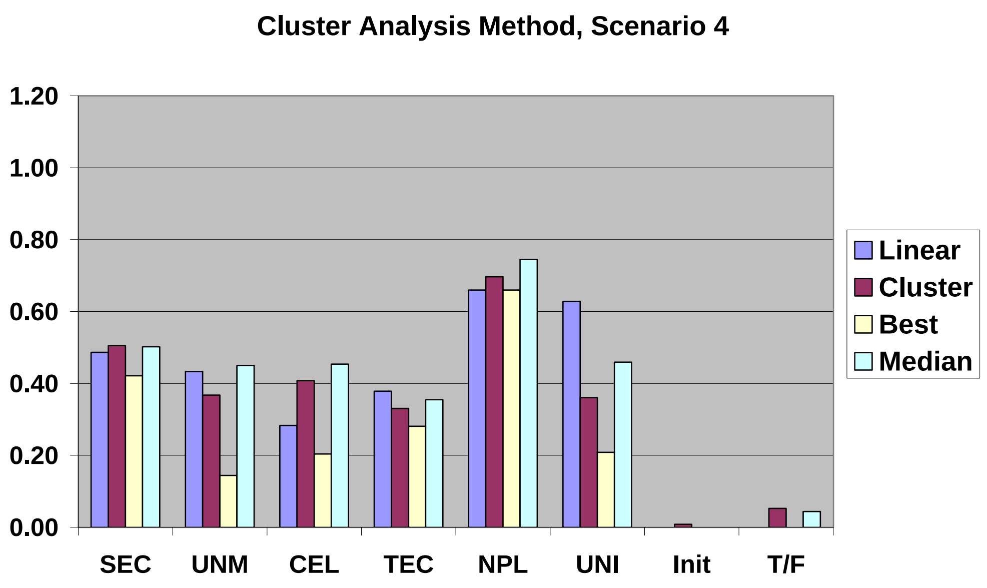


Figure 6.25: This figure shows 9 snapshots of a 101 frame animation. The animation is the result of applying the *Motion Cluster Analysis* method to scenario 4.

numerical results of the animation in regard to our measures are summarized in the following table:

Method	SEC	UNM	CEL	TEC	NPL	UNI	Init	T/F
Cluster	197	2.441E+08	1,909,777	1,160	127,250	97	220	0.6
Linear	190	2.877E+08	1,326,461	1,328	120,561	169	0	0.0
Best	164	9.562E+07	954,237	985	120,561	56	0	0
Median	196	2.991E+08	2,125,756	1,244	136,133	124	0	0.5



The *Cluster Analysis* method is not able to identify any significant motion clusters and the resulting animation is quite similar to the *Direct Linear Interpolation* method, the median method, and the best method.

Scenario 5

Figure 6.26 shows the result of applying the *Motion Cluster Analysis* method to scenario 5. The

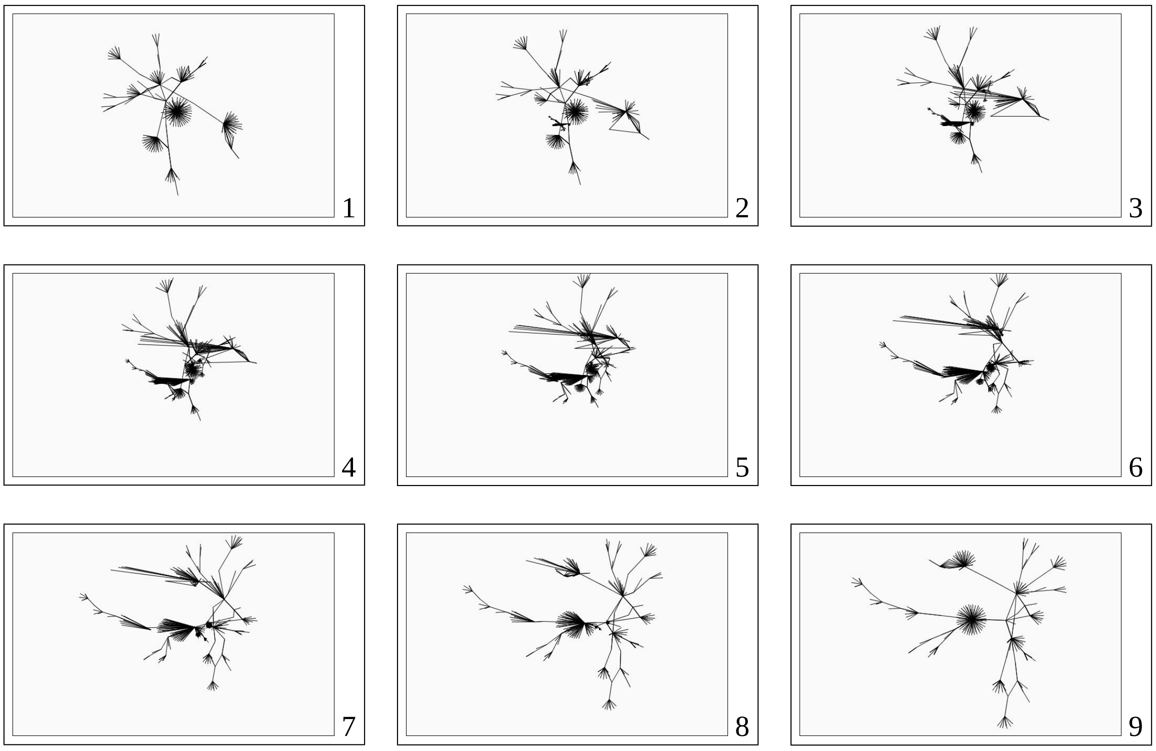
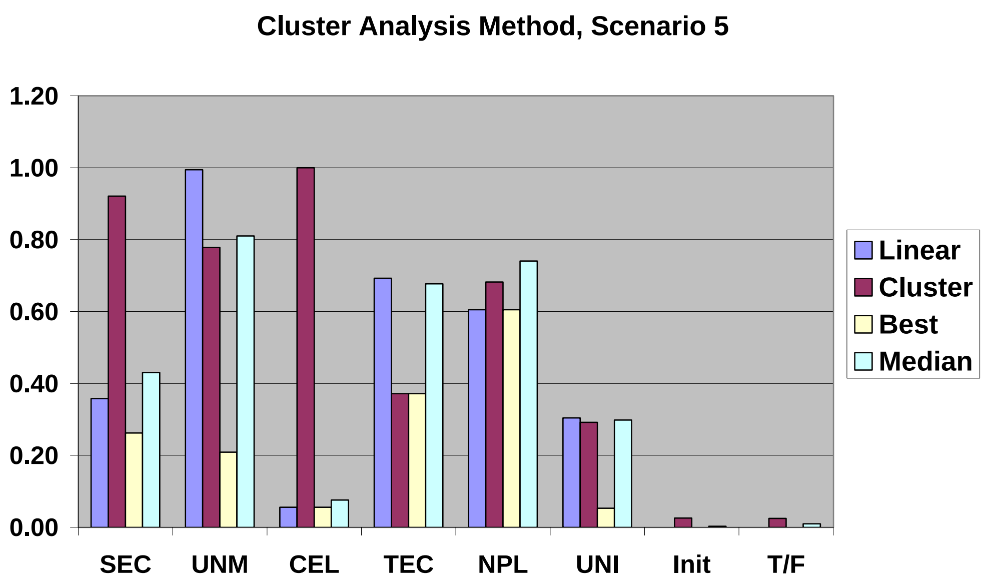


Figure 6.26: This figure shows 9 snapshots of a 101 frame animation. The animation is the result of applying the *Motion Cluster Analysis* method to scenario 5.

numerical results of the animation in regard to our measures are summarized in the following table:

Method	SEC	UNM	CEL	TEC	NPL	UNI	Init	T/F
Cluster	1,081	1.151E+10	43,154,023	8,438	673,484	266	550	2.9
Linear	420	1.470E+10	2,399,401	15,725	597,271	277	0	0.0
Best	308	3.088E+09	2,399,401	8,438	597,271	48	0	0
Median	505	1.198E+10	3,262,048	15,373	731,150	272	55	1.1



Again the *Cluster Analysis* method identifies several motion clusters with high confidence. However, the motion clusters are obviously not identical to the edge connectivity clusters of the graph. Although not disastrous, the animation nevertheless looks counter intuitive. The recorded measures reflect this by good values for some measures, but very bad values for *constant edge length* and *static edge crossings*.

Scenario 6

Figure 6.27 shows the result of applying the *Motion Cluster Analysis* method to scenario 6. The

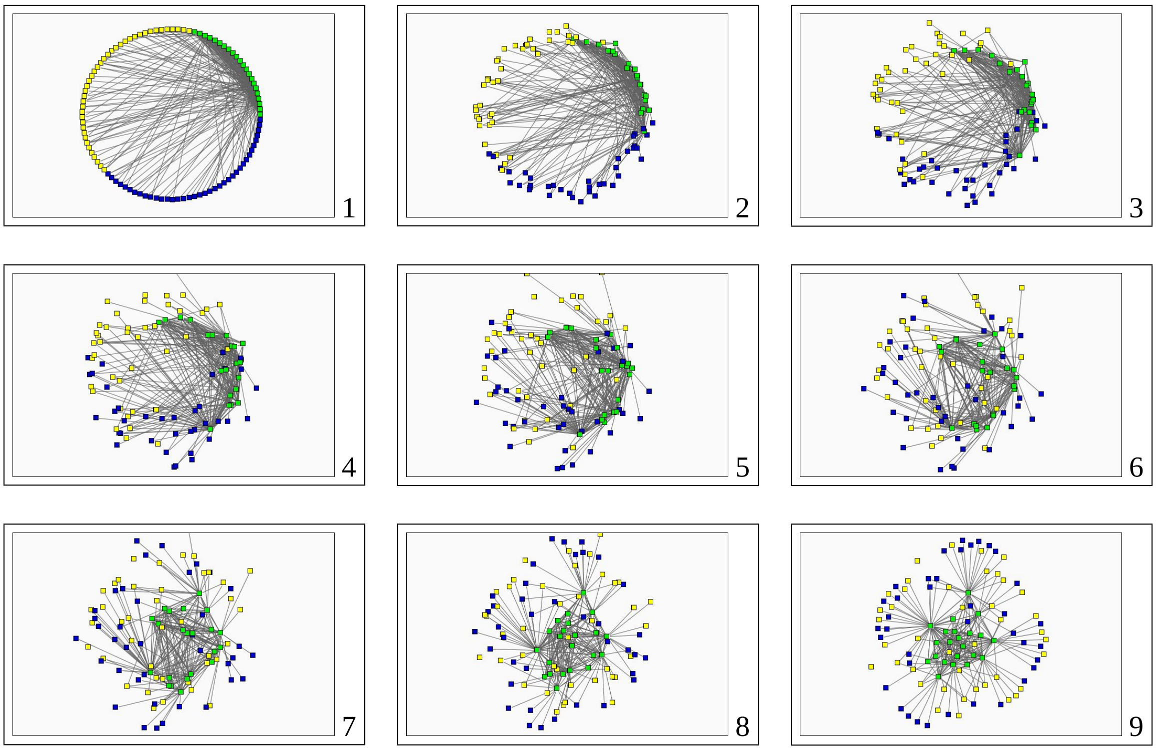
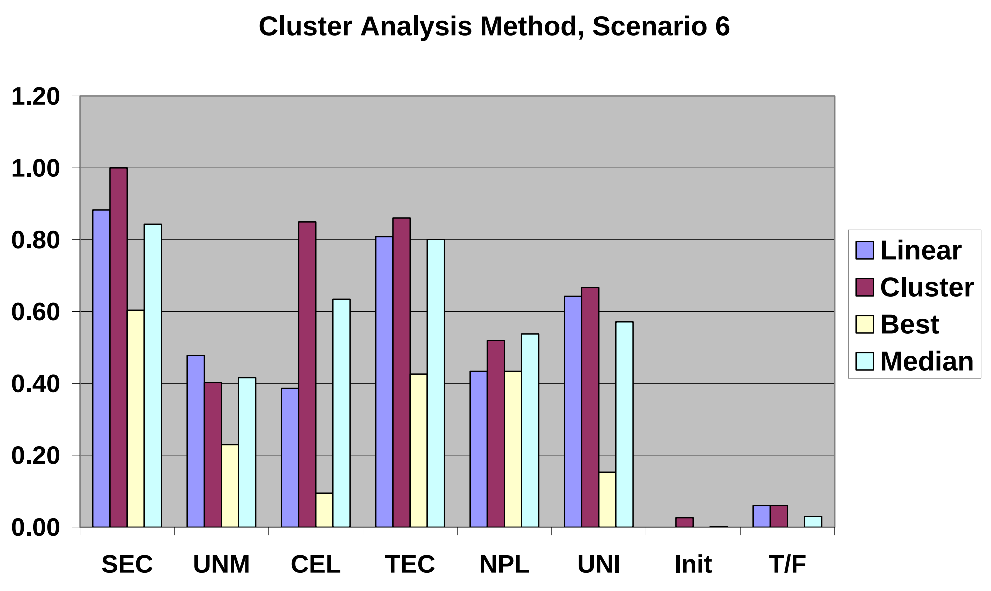


Figure 6.27: This figure shows 9 snapshots of a 101 frame animation. The animation is the result of applying the *Motion Cluster Analysis* method to scenario 6.

numerical results of the animation in regard to our measures are summarized in the following table:

Method	SEC	UNM	CEL	TEC	NPL	UNI	Init	T/F
Cluster	8,040	2.009E+08	9,184,635	16,768	88,536	192	330	0.5
Linear	7,095	2.385E+08	4,174,533	15,762	73,957	185	0	0.5
Best	4,853	1.147E+08	1,020,039	8,301	73,957	44	0	0
Median	6,777	2.077E+08	6,856,478	15,610	91,672	165	25	0.2



The results for scenario 6 and the following scenarios is consistent with the remarks for the previous scenarios.

Scenario 7

Figure 6.28 shows the result of applying the *Motion Cluster Analysis* method to scenario 7. The

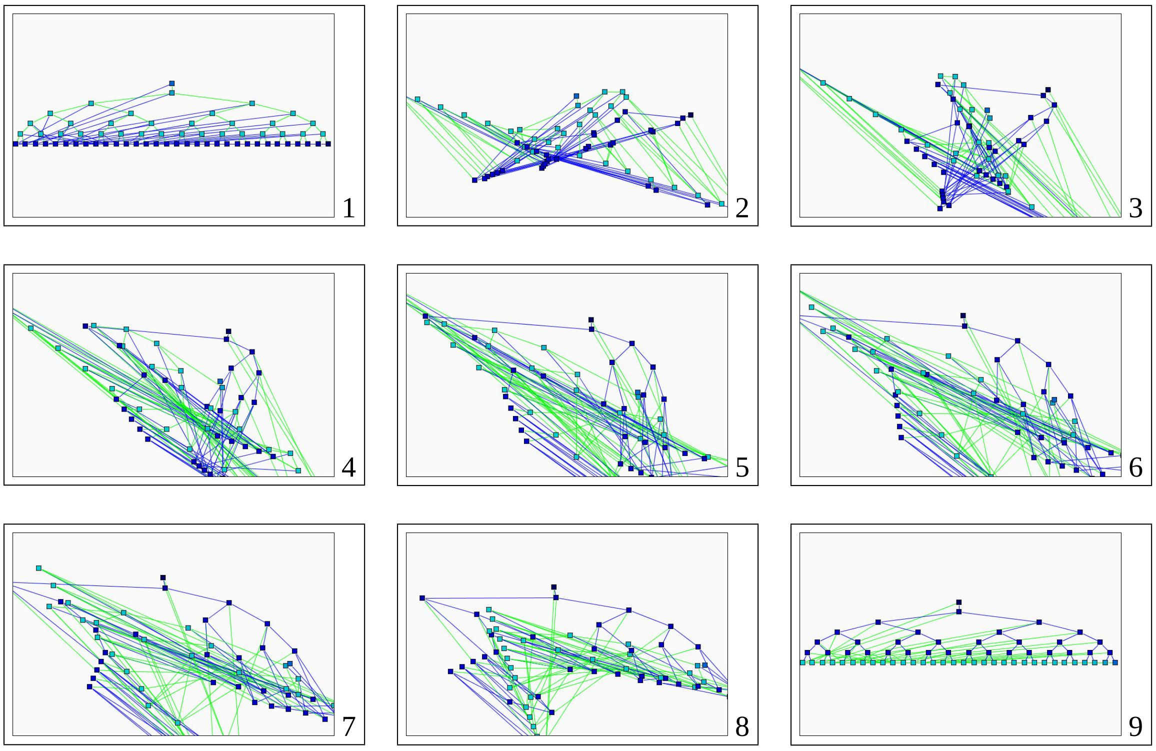
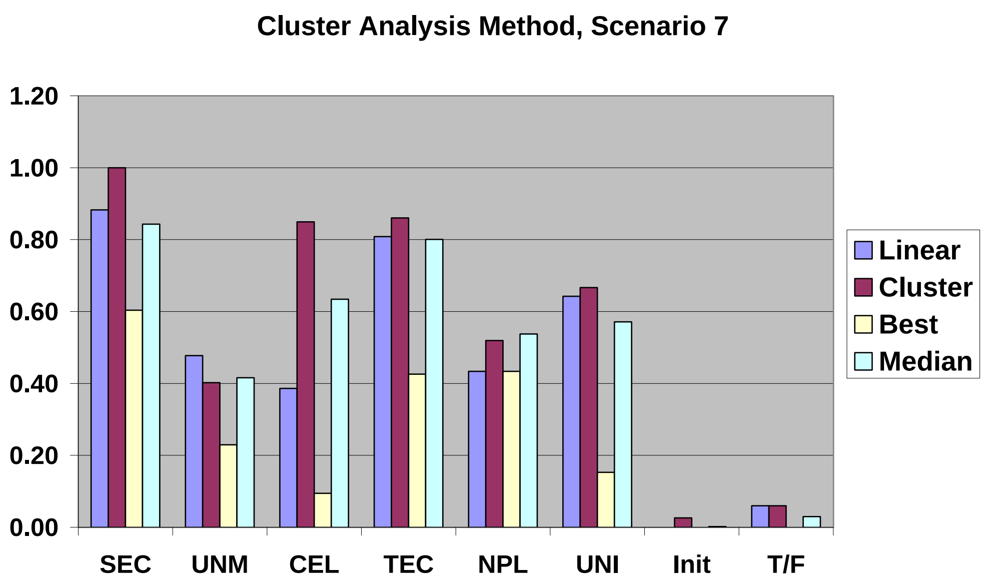


Figure 6.28: This figure shows 9 snapshots of a 101 frame animation. The animation is the result of applying the *Motion Cluster Analysis* method to scenario 7.

numerical results of the animation in regard to our measures are summarized in the following table:

Method	SEC	UNM	CEL	TEC	NPL	UNI	Init	T/F
Cluster	1,195	9.394E+07	11,209,526	4,916	50,338	197	220	2.3
Linear	1,011	4.858E+07	90,948	4,975	25,170	2,692	0	0.0
Best	454	1.841E+07	71,857	2,905	25,170	77	0	0
Median	696	4.262E+07	816,052	4,631	45,283	198	25	1.1



The results for scenario 7 are consistent with the remarks for the previous scenarios.

Scenario 8

Figure 6.29 shows the result of applying the *Motion Cluster Analysis* method to scenario 8. The

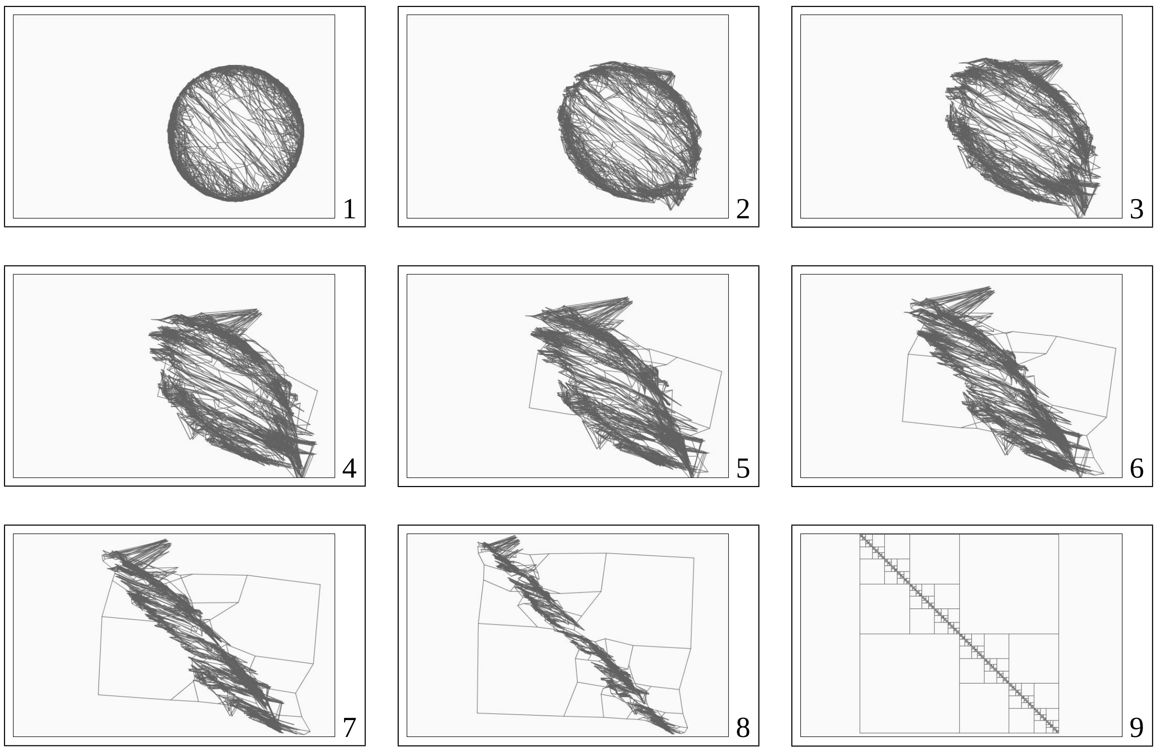
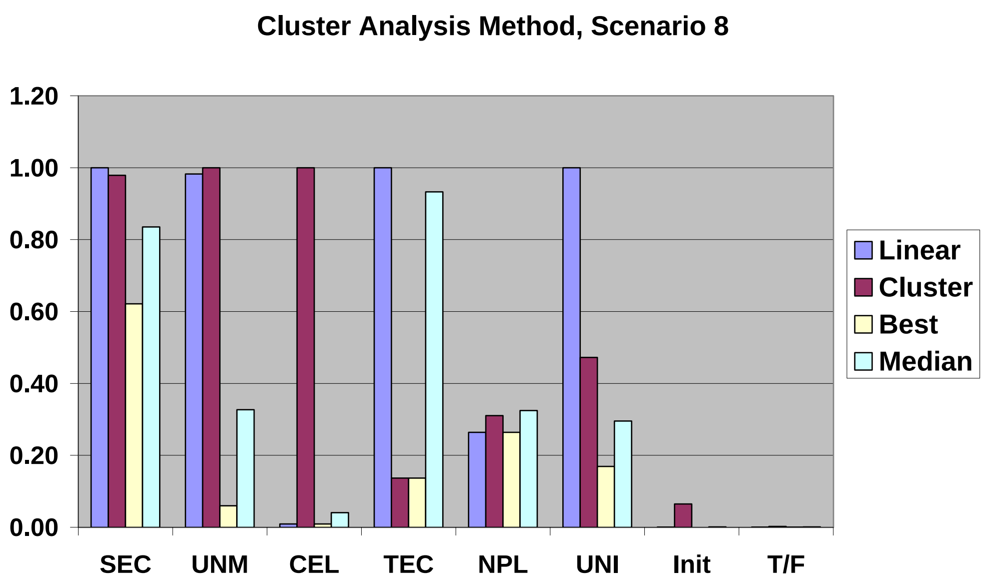


Figure 6.29: This figure shows 9 snapshots of a 101 frame animation. The animation is the result of applying the *Motion Cluster Analysis* method to scenario 8.

numerical results of the animation in regard to our measures are summarized in the following table:

Method	SEC	UNM	CEL	TEC	NPL	UNI	Init	T/F
Cluster	45,209	2.879E+09	453,584	289,169	341,731	3,507,042	5,660	32.7
Linear	46,185	2.829E+09	4,138	2,112,739	290,510	7,427,591	50	7.2
Best	28,706	1.718E+08	4,138	289,169	290,510	1,256,611	0	4
Median	38,574	9.407E+08	18,486	1,971,003	357,359	2,192,830	80	13.7



The results for scenario 8 are consistent with the remarks for the previous scenarios.

6.2.6 Summary

During our work on the *Cluster Analysis* method we found that the method shows a high degree of variation of results between different scenarios and even between different runs of the same scenario. It is therefore not possible to base a discussion of the *Cluster Analysis* method solely on the 10 benchmark runs which we presented in the previous section. The following summary is thus also based on results which we gained by applying the *Cluster Analysis* method to a much larger number of animation problems during our research. These scenarios and their results are not explicitly mentioned in this thesis, however.

Using k-means to compute the animation in many cases successfully identifies different types of transformations, even in the presence of ‘local’ noise. Convergence is very fast, and the animations may be viewed in real time.

However, the general k-means method is known to be very sensitive to the choice of partition with which it is initialized [60, page 277]. Its variants, including our method, are not well-equipped to discover or escape from poor clusterings that are nevertheless locally optimal. The sensitivity of

k-means variants to their initial partitions has been investigated extensively [3, 82, 118]. In general settings, failure of the k-means heuristic is more likely when clusters are difficult to distinguish, or when a cluster is split among many groups of the initial partition.

In the context of animations, when clusters are difficult to distinguish, their associated transformations are necessarily similar. Whether two similar clusters are declared to be separate or the same, the resulting animations will not differ greatly. However, the distribution of a single cluster among many groups is a potentially much more serious problem. Such a clustering has the effect of a subgraph transformation not being applied in the animation. Further, partitionings in which the motion clusters do not correspond to the edge connectivity clusters of the graph often result in counter-intuitive animations. Extending the clustering model to include edge-connectivity information may considerably increase the quality and stability of the *Cluster Analysis* method.

CHAPTER 7

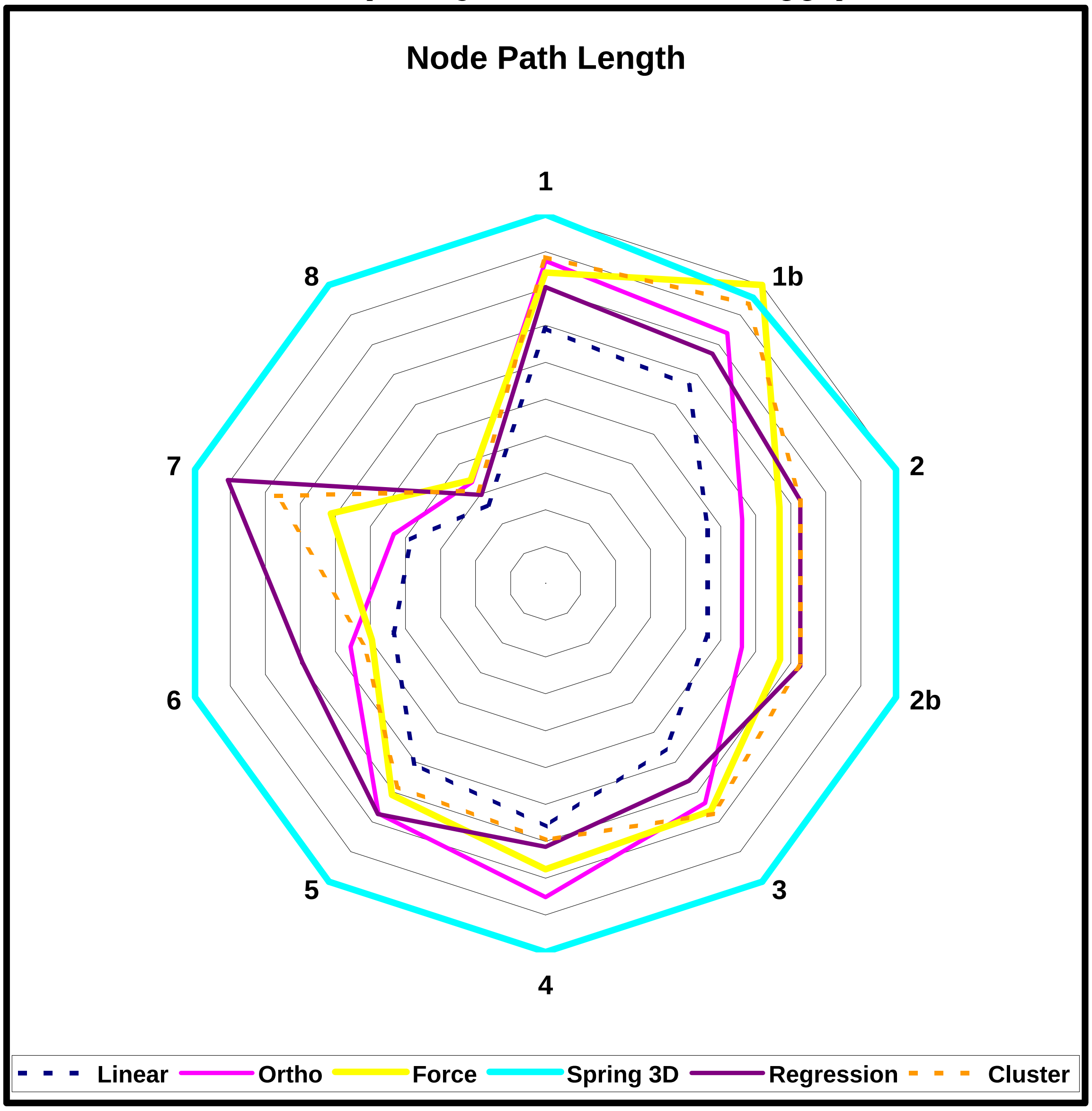
Conclusions

7.1 General Analysis of Animation Methods by Measure

In the previous chapters, we introduced several methods to autonomously compute animations between drawings of graphs. We discussed these methods analytically and applied them to our benchmark suit. The results of the benchmark tests were compared to the median and best result for each measure. So far, we have not directly compared animation methods with each other. We do this in the following sections. For each measure we compare how the different methods perform relative to each other over all benchmark scenarios. In this context we present a “radar” graphic for each measure which shows the results for all methods on the benchmark scenarios. The results are normalized to the radius of the radar. The exact numerical values can be found in appendix A. Graphics displaying the results for all measures and methods grouped by benchmark scenario are included in appendix B.

7.1.1 Node Path Length

The results for the measure *node path length* are shown in the following graphic:



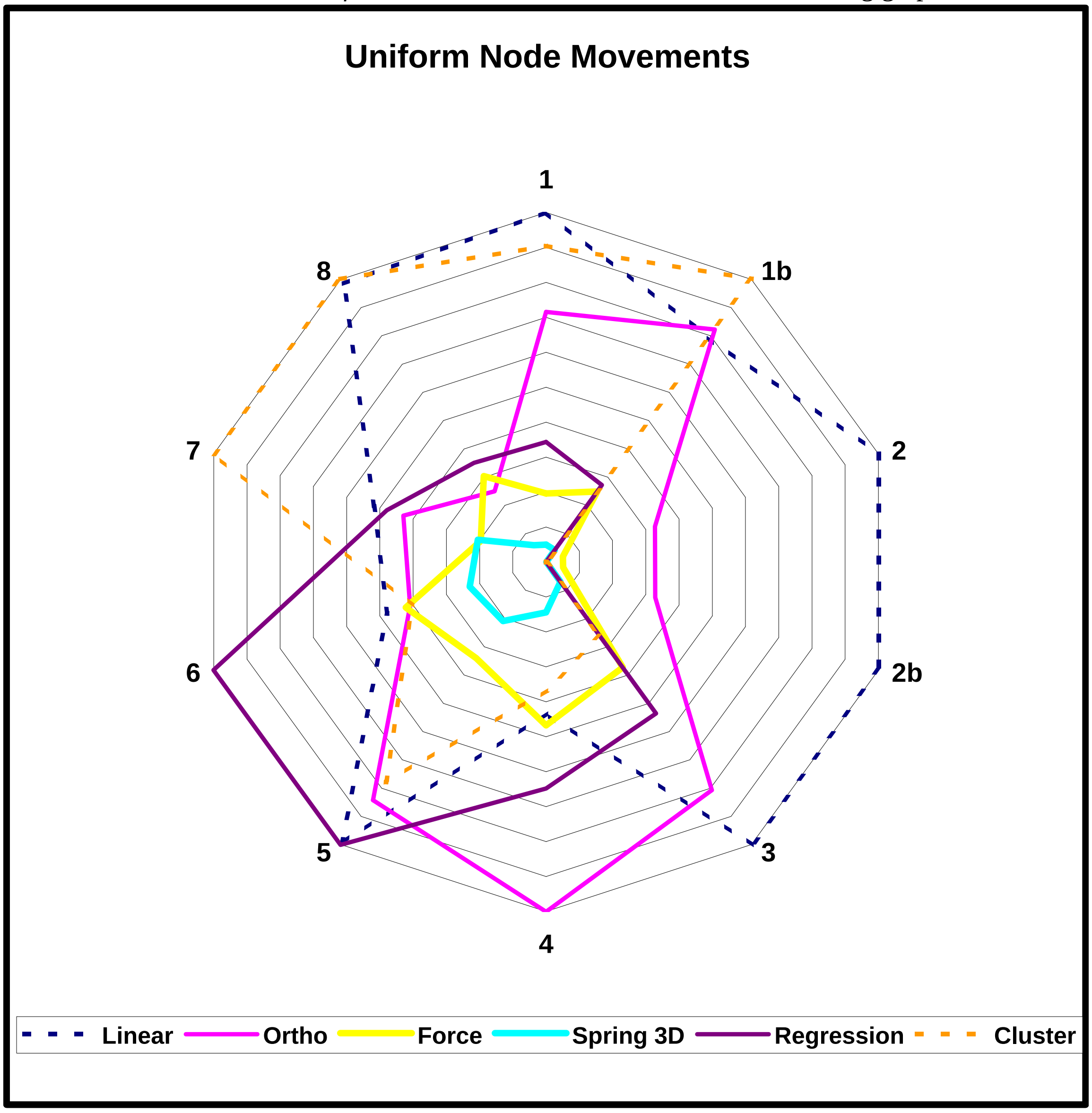
By design, the *direct linear interpolation* method achieves optimal values for *node path length*. Apart from scenario 1b, the *3D Spring Embedder* method performs worst in all benchmark scenarios. The artificial framework which is supposed to hold the graph together while computing the animation is obviously not strong enough to guarantee short node paths. The results are particularly bad for scenario 8. The question arises whether the node paths during *3D Spring Embedder* animations are "too long" or just "longer than the others". We do not have a formal criteria to answer this question and thus have to rely on the subjective impression of watching the animation (Figure 5.25 on page 198). In our opinion the node paths during this animation do not seem excessively long.

Most methods perform significantly worse than direct linear interpolation in scenario 7. On the other hand, we can see in the graphic for *unnecessary node intersections* on page 274 that *Direct Linear Interpolation* performs significantly worse than all other methods regarding *unnecessary node intersections* (UNI) in scenario 7. This indicates an essential trade off between *node path length* and *unnecessary node intersections* for this scenario. The topology of this graph changes significantly between the initial and target drawing. This means that the nodes have to negotiate their path to their respective target positions among many other nodes moving in different directions. Long node paths seem a natural consequence if node collisions are to be avoided. However, an optimal solution for this scenario is not known and we can therefore not exclude the possibility that there is a solution which achieves good values for all measures in scenario 7. For all other scenarios the values for all methods lie relatively close to each other.

To summarize, we observe that although the *direct linear interpolation* method achieves best values for *node path length* in all scenarios, the other animation methods also seem generally to be able to produce animations with good *node path length*.

7.1.2 Uniform Node Movement

The results for the measure *uniform node movement* are shown in the following graphic:



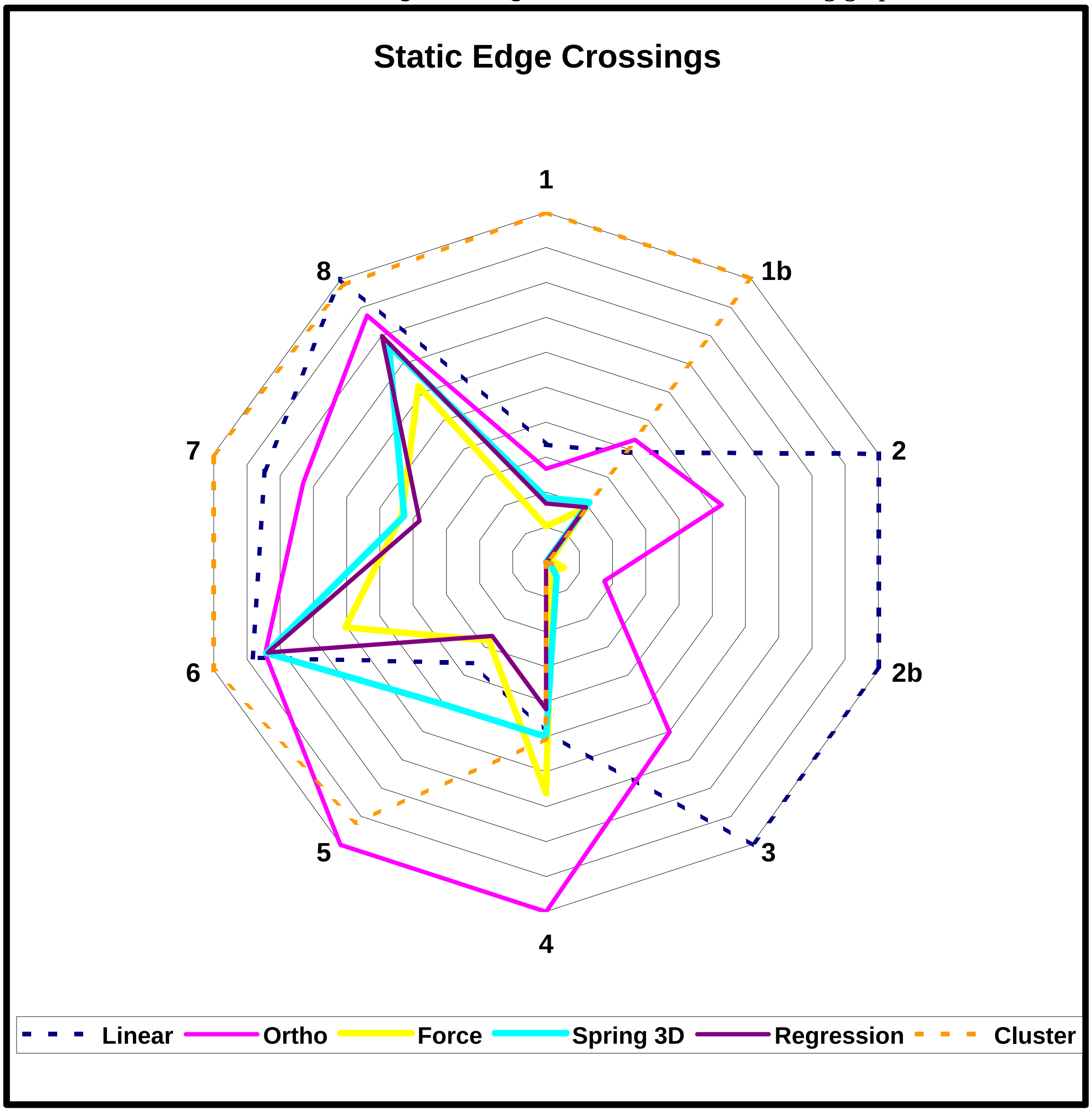
The *uniform node movement* measures how much the distance between each pair of nodes varies from their linearly interpolated distance during the animation. The *Force Directed* method and the *3D Spring Embedder* method try to optimize *uniform node movement* explicitly. The graphic shows that they achieve this goal better than any other method over all scenarios.

The *Linear Regression Analysis* method performs significantly better than *Direct Linear Interpolation* in all scenarios but scenario 6. Scenario 6 involves two different kind of motions. The green nodes move towards the center of the drawing while the blue and the yellow nodes remain on the periphery of the initial circle. As the *Linear Regression Analysis* method computes the "average" movement of all nodes, averaging the two different kinds of motion significantly decreases the quality of the computed motion. Although the *Cluster Analysis* method is able to cope with the different motions in scenario 6, it exhibits a significantly bad behavior in scenario 7. And indeed, the actual animation (Figure 6.28 on page 257) does not appear particularly convincing. The identified motion clusters look valid but do not follow aesthetically pleasing motion paths.

We further notice that the *Orthogonal Interpolation* method performs significantly worse than all other methods in scenario 4. It is not quite clear why this is the case. The main merit of the *Orthogonal Interpolation* method lies in the aesthetically pleasing three dimensional illusion which it generates. This illusion cannot be captured by our measures. Consequently, we cannot expect to be able to fully explain its behavior with our measures. We continue to include the results for the *Orthogonal Interpolation* method in our graphics, but, for the reasons given above, do not discuss them in the future.

7.1.3 Static Edge Crossings

The results for the measure *static edge crossings* are shown in the following graphic:

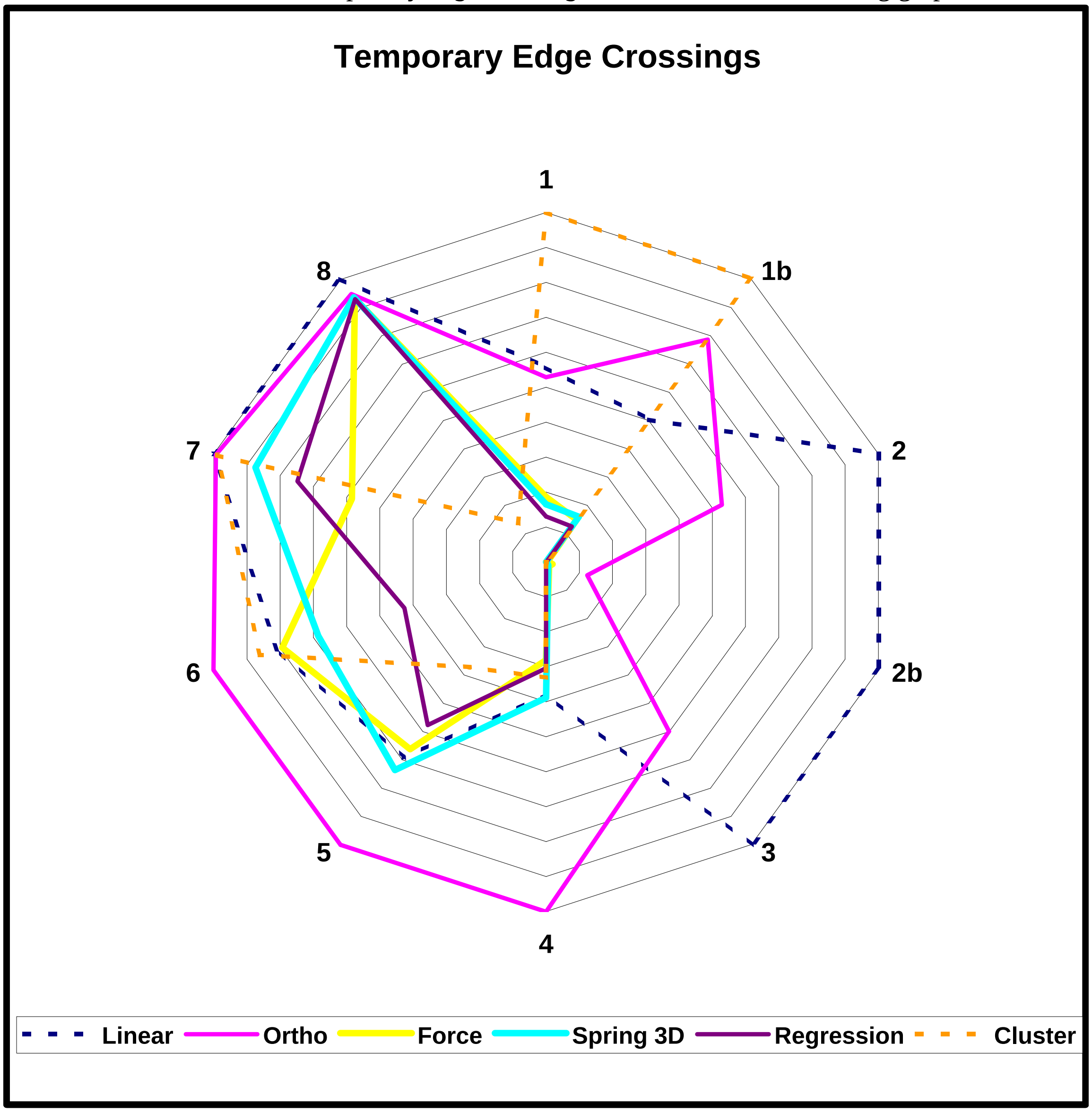


The *Linear Regression* method performs better than *Direct Linear Interpolation* in all scenarios. The *Force Directed* method and the *3D Spring Embedder* method perform better or almost as well as *Direct Linear Interpolation* in all scenarios.

The *Cluster Analysis* method achieves good values for the structured scenarios 2, 2b, and 3, as well as for scenario 4. However, the values for scenarios 1, 1b, and 5 are very bad. We have noted before that the corresponding animations for scenarios 1 and 1b give a rather poor aesthetic impression. Examining the actual animation reveals that, again, the *Cluster Analysis* method has identified motion clusters which do not correspond to edge-connectivity based clusters. Again, the motions appear valid motions, but counter our intuition.

7.1.4 Temporary Edge Crossings

The results for the measure *temporary edge crossings* are shown in the following graphic:



The *Linear Regression* method and the *Force Directed* method are more successful in avoiding temporary edge crossings than *Direct Linear Interpolation* in all scenarios.

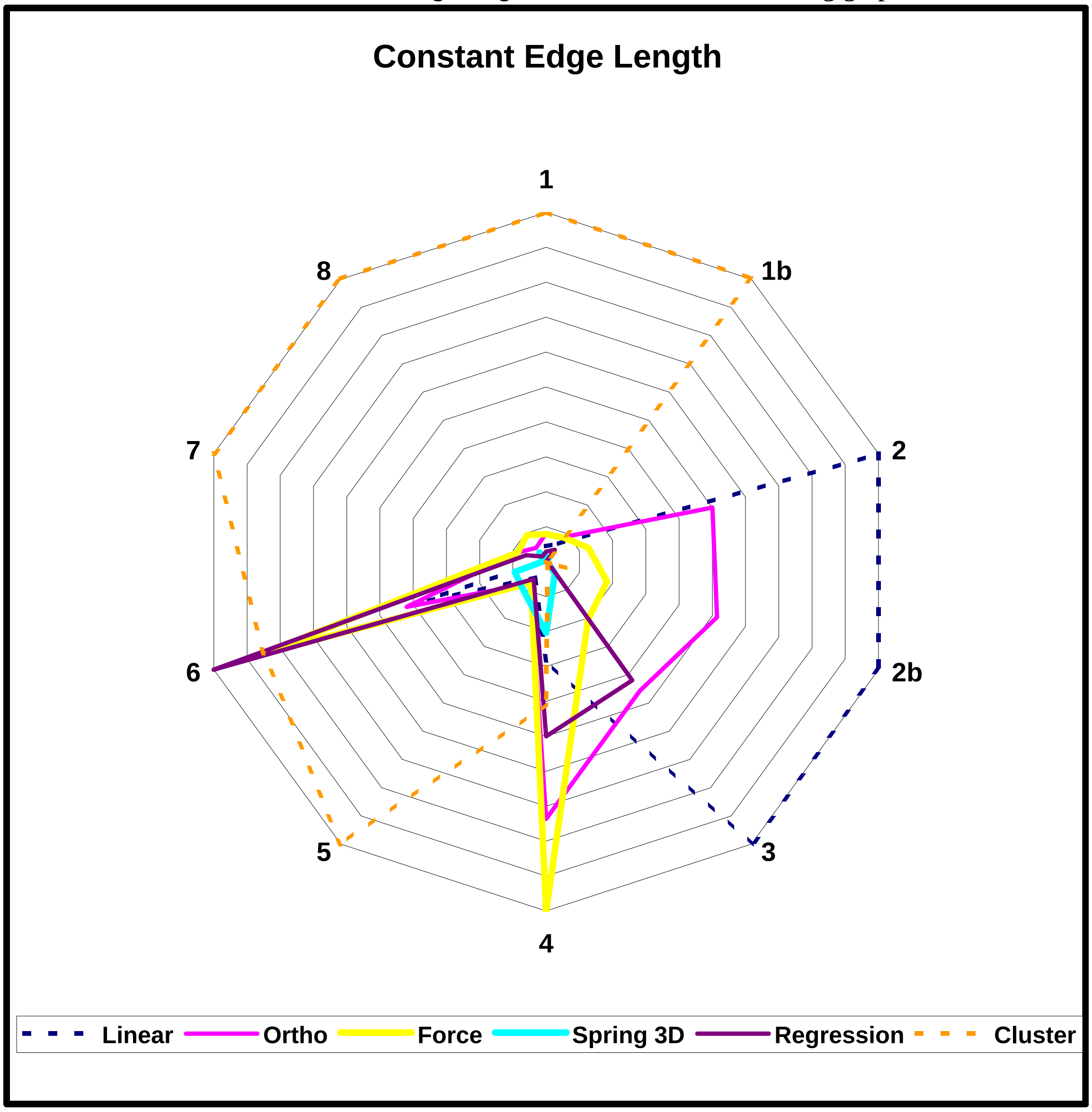
The *3D Spring Embedder* method performs better or almost as well as *Direct Linear Interpolation* in all scenarios.

The *Cluster Analysis* method achieves bad values in scenarios 1 and 1b, but good values otherwise. We note exceptionally good values for scenario 8. The animation itself, however, does not seem to reflect this good value in any obvious way. Subjectively it does not look significantly better than the result, for example, for the *Linear Regression* method.

All methods perform better than the *Linear Interpolation* method in most cases regarding *temporary edge crossings*. This is a strong indication that they are generally able to produce a topologically more stable animation than the *Linear Interpolation* method. We consider this fact significant with respect to the goal of helping the user maintain the mental map of the graph during the animation.

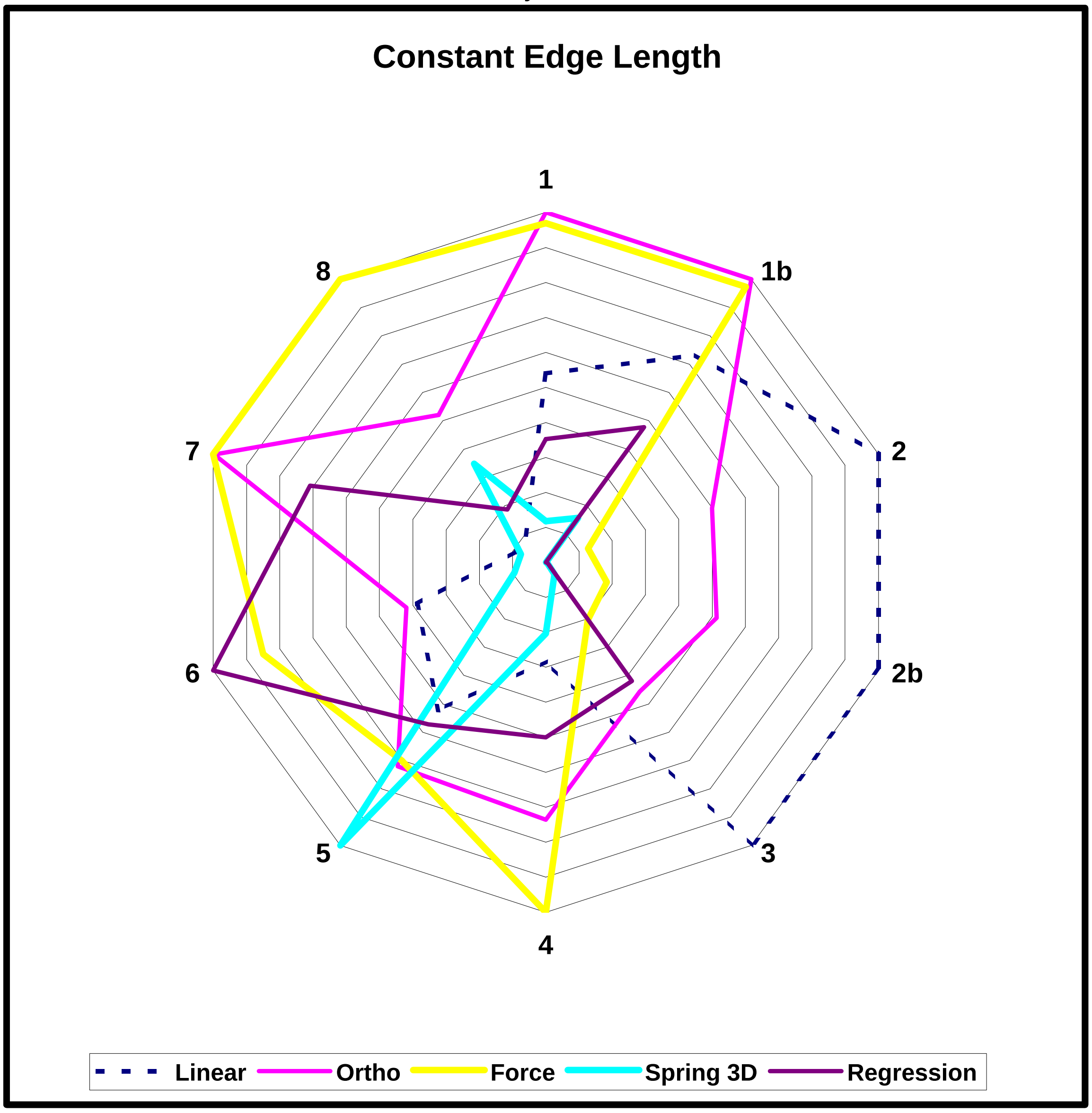
7.1.5 Constant Edge Length

The results for the measure *constant edge length* are shown in the following graphic:



The *Cluster Analysis* method performs well in the structured scenarios 2, 2b, and 3, as well as in scenario 4. The values for *constant edge length* are very bad in all other scenarios. The *Cluster Analysis* method tries to identify motion clusters. These clusters are not necessarily identical to clusters based on edge connectivity. As a result nodes which are connected by edges can be assigned to different motion clusters. These inter-cluster edges can become excessively distorted during the animation. Where motion clusters are similar to edge connectivity based clusters (Scenarios 2, 2b, 3), the *Cluster Analysis* method achieves good values for *constant edge length*.

The very bad results for the *Cluster Analysis* method skew the graphic for the scenarios 1, 1b, 5, 7, and 8. We therefore also give the following graphic of the *constant edge length* measure which does not contain the results for the *Cluster Analysis* method:



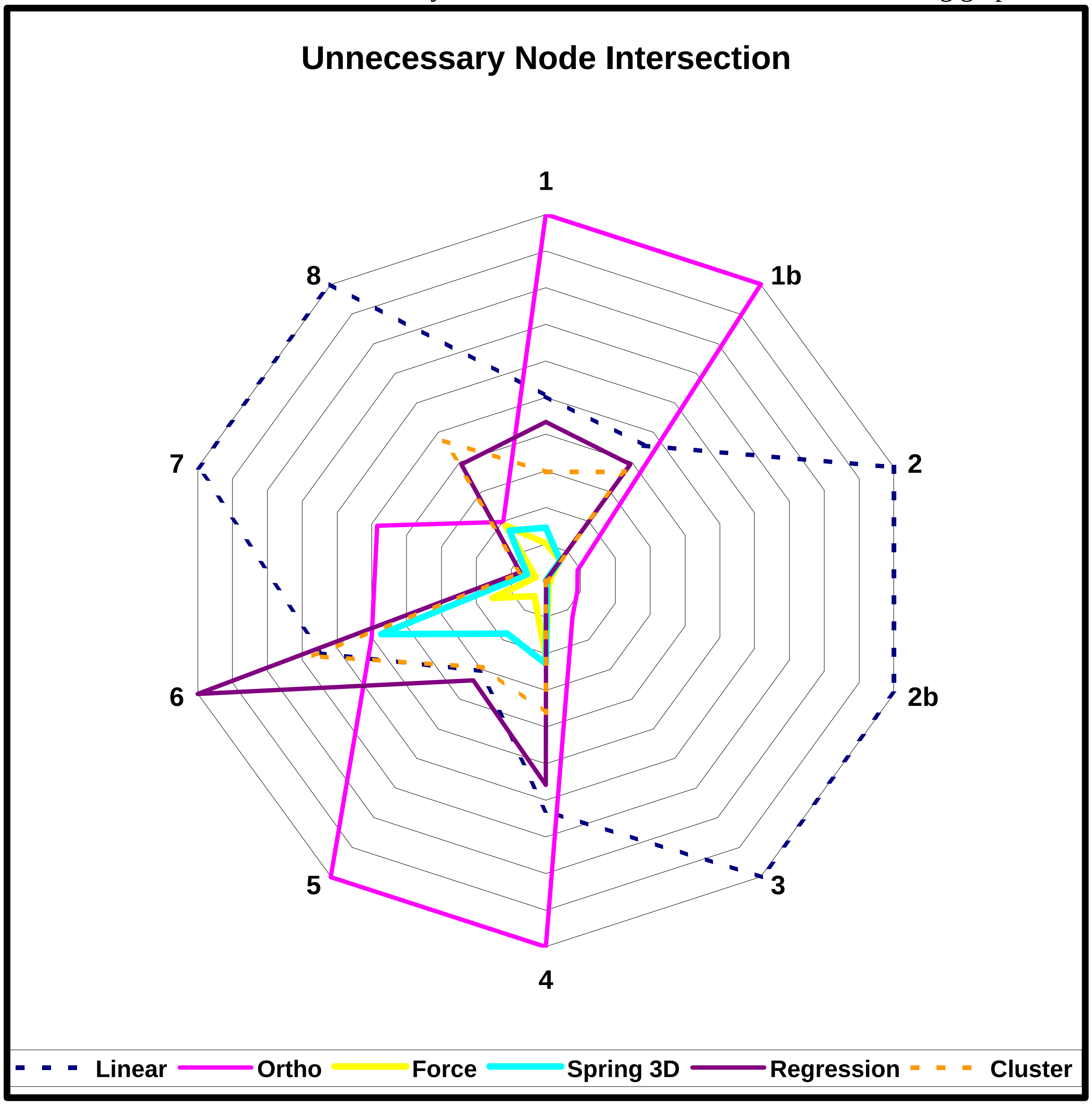
Ignoring the results for the *Cluster Analysis* method we notice that the *Direct Linear Interpolation* method generally achieves relatively good results in all non-structured scenarios. The *3D Spring Embedder* method is still able to outperform *Direct Linear Interpolation* in all scenarios but scenario 5 and 8. However, in both scenarios the *3D Spring Embedder* method achieves significantly better results for *uniform node movement* and *unnecessary node intersections*.

The *Force Directed* method achieves comparatively poor values for *constant edge length* in some scenarios. This seems counter-intuitive at first, as the *Force Directed* method tries to optimize *uniform node movement*, and *constant edge length* is equivalent to *uniform node movement* when only applied to a subset of all node pairs. The *Force Directed* method, therefore, indirectly also tries to optimize *constant edge length*. For most graphs the number of edges is linear to the number of nodes in the graph. Thus, in cases where there is an essential tradeoff between *constant edge length* and *uniform node movement* we can expect the forces which support *constant edge length* to be opposed by a quadratic number of forces which support *uniform node movement*. This results in bad values for *constant edge length* in these cases.

The *Linear Regression* method achieves good results in all scenarios but scenario 6 and 7. We previously discussed the general difficulties which the *Linear Regression* method has in computing a good animation for scenario 6. By watching the corresponding animation (Figure 5.10 on page 167) we can see that the animation contains a comparatively high skew component. This distorts the drawing and is ultimately to blame for the bad value for *constant edge length*. Whether the bad value for *constant edge length* is compensated by the increased amount of structure in the motion does not become obvious by watching it. It would be necessary to conduct HCI experiments to answer this question.

7.1.6 Unnecessary Node Intersections

The results for the measure *unnecessary node intersections* are shown in the following graphic:



The *Force Directed* method, the *3D Spring Embedder* method, the *Cluster Analysis* method, and the *Linear Regression Analysis* method achieve very good values in all scenarios. In 5 of the 10 scenarios the values are significantly better. These 5 scenarios include the structured scenarios (2,2b,3) and scenarios 7 and 8.

We already identified the good values for scenario 7 to be the result of an apparent trade-off between *node path length* and *unnecessary node intersections* in this scenario. As the *Direct Linear Interpolation* method, by definition, achieves optimal values for *node path length*, it is not surprising to observe that it gets rather bad results for *unnecessary node intersections*

We further notice that the *Linear Regression* method achieves comparatively bad results for scenario 6. While the green nodes move towards the center of the graph in this scenario, the other nodes remain on the periphery of the graph. As the *Linear Regression Analysis method* averages the motions, the non-green nodes are unnecessarily forced to partly move towards the center of the graph. This increases the chance of intersections.

7.2 Guidelines

The previous theoretical analysis and empirical evaluation of the proposed animation methods allows us to propose the following guidelines for the use of animation methods in real world applications.

We cannot identify an overall best method regarding all measures. Rather, we found that the proposed animation methods have individual advantages and disadvantages which makes the choice of which method to use highly application dependent. The following table shows relevant application parameters and the respective expected performance of the animation methods. The table and the subsequent discussion are based on the results of applying the animation methods to the benchmark suit[1], the intuitive aesthetic impression of watching the corresponding animations, and the accumulated experience which we gained by working with these animation methods during our research. The numbers in the table range from 1 to 5, with 1 representing very good, and 5 representing very bad.

Method \ Property	General	Structure[2]	Sub-graph structure	Graph Size
Linear	3 to 5	3 to 5	3 to 5	1
Force	2 to 4	2 to 4	2 to 4	5
Spring 3D	2 to 5	2 to 4	2 to 4	4
Regression	3 to 5	1	3 to 5	1
Cluster	4 to 5	1 to 3	1 to 4	1

In cases where results can differ significantly between different scenarios with the same property, we give a range of numbers representing the range of results which are most likely to occur. This means that the quality of the animation is determined by further, yet unknown, parameters in these

[1]See also appendix B

[2]We use the term “structure” to refer to transformations which can be approximated by an affine linear transformation. This especially includes transformations with a significant rotational component.

cases. For almost all parameters, it is possible to construct scenarios where a given method performs arbitrarily bad or good. We discuss the advantages and disadvantages of each method in the following paragraphs.

Direct Linear Interpolation The *Direct Linear Interpolation* method can produce good animations for simple changes between graph drawings. In trivial cases, the *Direct Linear Interpolation* method can even find an optimal solution. However, we found that in the majority of cases the resulting animations are rather poor.

Orthogonal Interpolation The *Orthogonal Interpolation* method moves the nodes of the graph in parallel to the y-axis and subsequently in parallel to the x-axis. This induces the illusion of a rotating rigid three-dimensional object. The animation is perceived as aesthetically very pleasing. It is doubtful, however, whether this kind of animation is actually increasing the users ability in maintaining or adjusting the mental map. We have seen in benchmark scenario 5 (Page 137) that, especially for changing graphs, the illusion of a constant three-dimensional object can be counter-intuitive.

Force Directed The *Force Directed* method produces good animations in almost all cases. It does not quite reach the same quality as specialized methods for certain specially structured scenarios, but still tends to produce good animations in these cases. The *Force Directed* method has several drawbacks, however. The most severe is the high computational cost. As the *Force Directed* method takes $O(n^2)$ time to compute each frame, its application is limited to small graphs or non-interactive systems. Further, the actual motion paths are only implicitly given by a set of non-linear equations. This set is not analytically solvable. As a consequence, the *Force Directed* method cannot provide random access to a given frame, does not allow reversing the direction of the animation[3], and makes it impossible to effectively control the speed of the animation.

3D Spring Embedder The *3D Spring Embedder* method produces good animations for most scenarios. Like the *Force Directed* method, it does not quite reach the same quality as specialized methods for certain specially structured scenarios, but still tends to produce good animations in these cases. The quality of the animations generated by the *3D Spring Embedder* method generally tends to be slightly less than for the *Force Directed* method. The *3D Spring Embedder* method is

[3]Unless all previous positions are explicitly recorded.

also more likely to produce bad animations than the *Force Directed* method, as can for example be seen in scenario 7 on page 198. In this scenario, the *3D Spring Embedder* method is not able to find a good local optimum for the animation framework. The resulting animation is quite inconsistent and unstructured. The *3D Spring Embedder* method has several advantages over the *Force Directed* method, however. As the whole animation is computed before the first frame is displayed, the *3D Spring Embedder* method can render each frame in linear time, provide random access to any frame, reverse the direction of the animation, and allows efficient control over the speed of the animation. The biggest drawback of the *3D Spring Embedder* method is its high initialization cost. Although it only has $O(n \log n)$ time complexity, a high constant factor almost always results in a noticeable gap before the animation starts on currently available computer systems. This gap can extend to several minutes for large graphs in certain scenarios. On current computer systems, the use of the *3D Spring Embedder* method is therefore limited to medium sized scenarios, non-interactive systems, or applications where the computational cost for computing the changes to the graph drawing outweigh the cost of initializing the animation.

Linear Regression Analysis The *Linear Regression Analysis* method produces good results in many cases. It is designed to identify and animate changes between drawings of graphs which can be described by an affine linear transformation. These especially include transformations with a significant rotational component. The *Linear Regression Analysis* method achieves very good results in these scenarios. For general scenarios, the *Linear Regression Analysis* method produces results which are at least as good as the *Direct Linear Interpolation* method for almost all cases. The *Linear Regression Analysis* method averages the overall motion of all nodes of the graph. As a consequence it achieves rather poor results if sub-graphs show different kinds of motions, as we have seen in figure 6.16 on pages 234. In these cases it sometimes even produces worse animations than the *Direct Linear Interpolation* method. We can therefore recommend the *Linear Regression Analysis* method as a general substitute for *Direct Linear Interpolation* with the exception of scenarios where distinctive and significantly different motion clusters are to be expected. When choosing between using the *Linear Regression Analysis* method and one of the force directed methods, time is the determining factor. Apart from scenarios in which the transformation can be approximated by an affine linear transformation of the graph, the force directed methods generally produce the better animation. However, they take significantly longer to compute an animation. For interactive systems the longer computation time may not be acceptable.

The efficiently computable r^2 value for the regression analysis can be taken into account when deciding whether to use the *Linear Regression Analysis* method for a given scenario. Although we were not able to identify a clear threshold, in our experience r^2 values of more than 70% generally indicate a good result for the *Linear Regression Analysis* method.

Cluster Analysis We designed the *Cluster Analysis* method to overcome the inherent problem of the *Linear Regression Analysis* method in handling differently behaving sub-graphs. As scenario 3 on page 249 shows, the *Cluster Analysis* method is able to successfully identify different clusters of structured motions. The *Cluster Analysis* method is also able to handle scenarios with only one cluster (scenario 2 on page 245), as well as the presence of noise (scenario 2b on page 245). For general graphs, however, the results are often disappointingly poor. Although the *Cluster Analysis* method is often able to identify valid motion clusters, these clusters, in many cases, appear counter-intuitive. The fact that the *Cluster Analysis* method only considers different kind of motions for clustering and completely ignores edge-connectivity based clusters in the graph can be identified as a reason for this effect.

7.3 Future Work

In this thesis we introduced different approaches to the problem of computing good animations between given initial and target drawings of graphs. As the first systematic work in this area we focused our research on an initial exploration of the solution space of the graph animation problem. As a consequence, we had to neglect other interesting aspects and thus, many possible directions for future research present themselves.

Criteria for good animations

We evaluated the proposed animation methods according to the measures and benchmark scenarios which we introduced in chapter 3. The empirical evaluation has shown that the set of measures is strongly related to our intuitive perception of the quality of graph animations. Further studies are necessary, however, to formally validate this claim.

Thorough HCI experiments have to be conducted to validate and rank the set of animation criteria. It is likely that further criteria, which determine the quality of a graph animation, may be identified. A complete and weighted set of criteria for the perceptual quality of graph animations would be an extremely valuable contribution to this area.

Special cases of structured transformations

When designing animation methods we focussed on general or simply structured transformations between graph drawings. Many special cases exist where our animation methods produce comparatively poor results, while humans are often able to instantly devise a good animation strategy for these cases. Figure 7.1 shows such a case. It is easy for humans to see that uncurling the spiral from figure 7.1.1 would be a good and simple way to animate this scenario. Our algorithms, however, are not able to find this animation.

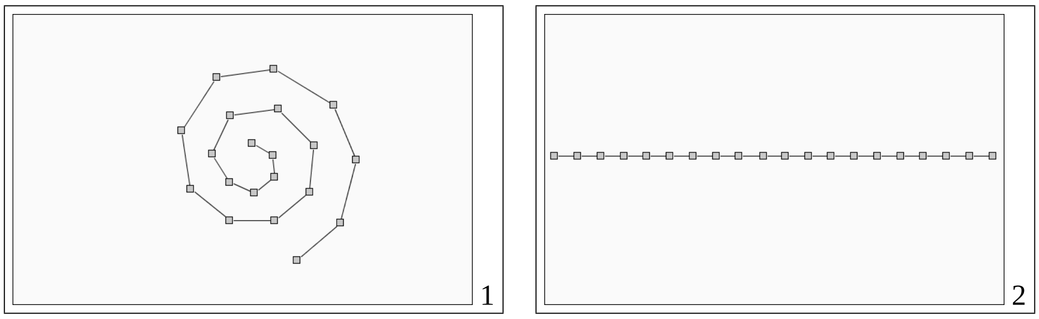


Figure 7.1: Example of a special case scenario which is challenging for graph animation algorithms. Although humans can devise a good animation strategy immediately (uncurling of the spiral), our algorithms are not able to find it.

Possible improvements of proposed methods

Some of the methods proposed in this thesis have potential for further improvement.

Force Directed Method The quality of animations which are generated by the *Force Directed* method strongly depend on an appropriate setting of the weights of the individual forces. For our experiments we used a fixed set of weights which we found to work well in general cases. Methods to automatically and efficiently determine good weight ratios for given scenarios could be able to improve the *Force Directed* method. We further noticed that the *Force Directed* method tends to achieve comparatively poor values for *constant edge length* (CEL). It might be possible to overcome this problem by introducing additional forces.

3D Spring Embedder Method The same remarks apply to the *3D Spring Embedder* method. Better weight ratios and additional forces to overcome specific shortcomings could improve the quality of the *3D Spring Embedder* method significantly. Although the *3D Spring Embedder* method is fast in theory, taking $O(n \log n)$ computation time, we notice a high constant factor which limits its application on current computer systems. We use a straightforward modification of the GEM spring embedder algorithm [42] to simulate the forces. An approach which specializes on the specific properties of the *3D Spring Embedder* method might be able to decrease computation time. Another shortcoming of the current implementation of the *3D Spring Embedder* method is that node paths are interpolated on a straight line between the virtual key frames. This can result in abrupt changes of direction and thus a non-smooth animation. Even in cases where the *3D Spring Embedder* method finds the optimal solution to a given animation problem such behavior is possible. In the case of a graph performing a rotation, the node paths, as computed by the *3D Spring Embedder* method, are piecewise straight line segments. For large graphs the change of direction at each virtual key frame is noticeable. Figure 5.19 on page 186 shows such behavior to some degree. This effect could be lessened by increasing the number of virtual key frames. However, the corresponding increase in computation time, although it only adds to the constant factor, would be too high. Another approach could be to interpolate the node paths using splines. Especially the use of NURBS could yield a considerable improvement.

Cluster Analysis Method We have seen that the *Cluster Analysis* method performs very well in structured scenarios, but has problems computing a good animation for non-specific changes between graph drawings. Improved clustering methods might be able to overcome this problem. Especially taking the edge-connectivity of the underlying graph into account might boost the performance of the *Cluster Analysis* method. It may also be possible to try to extend the set of recognized kinds of motions of sub graphs.

Choosing the appropriate method

Our benchmark tests revealed that our animation methods perform better for some scenarios than for others. Especially, that there is no single method which performs better than all other methods all the time. We were able to identify some criteria which are related to the expected quality of an animation, such as the r^2 value for the *Linear Regression* method and the *Cluster Analysis* method, or the final energy of the spring system in the *3D Spring Embedder* method. However, the significance of these criteria and usable threshold values are, so far, unknown. Finding such criteria would be a valuable contribution.

Bibliography

[1] Adelson, E., “Mechanisms for motion perception,” *Optics and Photonics News*, Vol. August, 1991, pp. 24–30.

[2] Akima, H., “A new method of interpolation and smooth curve fitting based on local procedures,” *Journal of Assc. for Comp. Mach.*, Vol. 17, No. 4, 1970, pp. 589–602.

[3] Aldenderfer, M. S. and Blashfield, R. K., *Cluster Analysis*, Sage Publications, Beverly Hills, USA, 1984.

[4] Baecker, R. M., *Interactive Computer-Mediated Animation*, Ph.D. thesis, Dept. Electrical Engineering, MIT, Cambridge, 1969.

[5] Barr, A. H., Currin, B., Gabriel, S., and Hughes, J. F., “Smooth interpolation of orientations with angular velocity constraints using quaternions,” *Computer Graphics*, Vol. 26, No. 2, 1992, pp. 313–320.

[6] Bartley, H. S., *Introduction to Perception*, Harper & Row, Publishers, Inc., 1980.

[7] Battista, G. D., Eades, P., Tamassia, R., and Tollis, I., “Algorithms for drawing graphs: An annotated bibliography,” *Comp. Geometry: Theory and Applications*, 1994, pp. 235–282.

[8] Battista, G. D., Eades, P., Tamassia, R., and Tollis, I. G., *Graph drawing: algorithms for the visualization of graphs*, Prentice-Hall Inc., 1999.

[9] Becker, R. A., Eick, S. G., and Wilks, A. R., “Visualising network data,” *IEEE Transactions on Visualisation and Computer Graphics*, Vol. 1 (1 March), 1995, pp. 16–28.

[10] Bederson, B. and Boltman, A., “Does animation help users build mental maps of spatial information?” *Info Vis ’99 Proceedings*, IEEE, 1999, pp. 28–35.

[11] Beier, T. and Neely, S., “Feature-based image metamorphosis,” *Computer Graphics*, Vol. 26, No. 2, 1992, pp. 35–42.

[12] Bendazzi, G., *Cartoons: One Hundred Years of Cinema Animation*, John Libbey, London, 1994.

[13] Bertault, F., “A force-directed algorithm that preserves edge-crossing properties,” *Information Processing Letters*, Vol. 74, No. 1–2, 2000, pp. 7–13.

[14] Brandenburg, F. J., Jünger, M., Marks, J., Mutzel, P., and Schreiber, F., “Graph Drawing Contest Report,” *Proceedings of the 7th international Symposium 1999, LNCS*, 1999, pp. 400–415.

[15] Bronstein, I. and et. al., K. S., *Taschenbuch der Mathematik*, Harri Deutsch Verlag, Frankfurt am Main, 200.

[16] Brown, M., “Zeus: A system for algorithm animation and multi-view editing,” *1991 IEEE Workshop on Visual Languages*, Vol. IEEE Computer Society Press, Kobe, Japan, 1991, pp. 4–9.

[17] Brown, M. H., “Exploring Algorithms Using Balsa-II,” *IEEE Computer*, Vol. 21, No. 5, 1988, pp. 14–36.

[18] Brown, M. H., “Perspectives on algorithm animation,” *Proceedings of the ACM SIGCHI ’88 Conference on Human Factors in Computing Systems*, May 1988, pp. 33–38.

[19] Burtnyk, N. and Wein, M., “Computer generated keyframe animation,” *Journal of the Society of Motion Picture and Television Engineers*, Vol. 80, No. 3, March 1971, pp. 149–153.

[20] Burtnyk, N. and Wein, M., “Interactive skeleton techniques for enhancing motion dynamics in key frame animation,” *Communications of the ACM*, Vol. 19, No. 10, 1976, pp. 564–569.

[21] Chernoff, H., “The use of faces to represent points in k-dimensional space graphically,” *Journal of the American Statistical Asoociation*, 1973, pp. 361–368.

[22] Cleveland, W., *The Elements of Graphing Data*, Hobart Press, Summit, New Jersey, 1994.

[23] Connelly, R., “Rigidity,” *Handbook of Convex Geometry*, Vol. A, 1993, pp. 223–271.

[24] Crafton, D., *Before Mickey: The Animated Film 1898-1928*, MA: MIT, Cambridge, 1987.

[25] Dam, E., Koch, M., and Lillholm, M., "Quaternions, interpolation and animation," Tech. Rep. DIKU-TR-98/5, Department of Computer Science, University of Copenhagen, DK-2100 Kbh, 1998.

[26] de Berg, M., van Kreveld, M., and Overmars, M., *Computational Geometry: Algorithms and Applications*, chap. 9, Springer Verlag, 2nd ed., 1998, pp. 188–200.

[27] Denis, M., *Image and Cognition*, Harvester Wheatsheaf, New York, 1991.

[28] Donskoy, M. and Kaptelinin, V., "Window navigation with and without animation: A comparison of scroll bars, zoom, and fisheye view," *In Proceedings of Extended Abstracts of Human Factors in Computing Systems (CHI 97)*, ACM Press, `http://www1.acm.org/sigs/sigchi/chi97/proceedings/short-talk/mdo.htm`, 1997, pp. 279–280.

[29] Eades, P., "A heuristic for graph drawing." *Congressus Numerantium.*, Vol. 42, 1984, pp. pp. 149–160.

[30] Eades, P. and Kelly, D., "Heuristics for reducing crossings in 2-layered networks," *Ars Combinatoria*, Vol. 21, 1986, pp. 89–98.

[31] Eades, P., Lai, W., Misue, K., and Sugiyama, K., "Preserving the mental map of a diagram," Tech. Rep. IIAS-RR-91-16E, Fujitsu Laboratories Ltd., 140 Miyamoto, Numazu-shi, Shizuoka 410-03, Japan, 1991.

[32] Eades, P., Lai, W., Misue, K., and Sugiyama, K., "Layout adjustment and the mental map," *Journal of Visual Languages and Computing*, Vol. 6, 1995, pp. 183 – 210.

[33] Eades, P. and Sugiyama, K., "How to draw a directed graph," *J. of Information Processing*, Vol. 13, 1990, pp. 424–437.

[34] Eades, P. and Zhang, K., *Software Visualization*, World Scientific, 1996.

[35] Edwards, D. and J.M.Graetz, "PDP-1 plays at spacewar," *Decuscope*, Vol. 1, No. 1, April 1962, pp. 2–4.

[36] Eloise H. Carlton, R. N. S., "Psychologically simple motions and geodesic paths," *Journal of Mathematical Psychology*, Vol. 34, No. 2, 1990, pp. 127–228.

[37] Fairchild, K. M., Poltrock, S., and Furnas, G. W., "Semnet: Three-dimensional graphic representations or large knowledge bases." *In R. Guidon (ed.) Cognitive Science and Its Applications for Human-Computer Interaction*, 1988, pp. 201–233.

[38] Feiner, S. and Beshers, C., "Worlds within Worlds: Metaphors for Exploring n-Dimensional Virtual Worlds," *in Proceedings of the ACM SIGGRAPH Symposium on User Interface Software and Technology*, 1990, pp. 76–82.

[39] Filliater, J., "Shape blending," `http://www.lems.brown.edu/vision/courses/image-processing/Projects-2000/ShapeBlending/ShapeBlending_files/frame.htm`, December 2000.

[40] Fleischer, R., "Algorithm animation," Seminar at the Information Visualization Group, Basser Department of Computer Science, The University of Syndey, August, 6th 2001.

[41] Forrest, A. R., "The twisted cubic curve: A computer-aided geometric design approach," *Computer Aided Design*, Vol. 12, No. 4, July 1980, pp. 165–172.

[42] Frick, Ludwig, and Mehldau, "A fast adaptive layout algorithm for undirected graphs." *Proc. of the DIMACS International Workshop on Graph Drawing (GD'94)*, 1994, p. .

[43] Friedrich, C., "The ffGraph library," Tech. Rep. 9520, Universität Passau, Dezember 1995.

[44] Friedrich, C. and Eades, P., "The Marey graph animation tool," *Proc. of the 8th Internat. Symposium on Graph Drawing (GD'2000)*, 2000, pp. 396–406.

[45] Friedrich, C. and Eades, P., "Graph drawing in motion," *to appear in Journal of Graph Algorithms and Applications*, 2002.

[46] Friedrich, C. and Houle, M., "Graph Drawing in Motion II," *to appear in Proc. of the 9th Internat. Symposium on Graph Drawing (GD2001)*, LNCS, Springer Verlag, 2001.

[47] Fruchterman, T. and Reingold, E., "Graph drawing by force-directed placement," *Software-Practice and Experience*, Vol. 21, No. 11, 1991, pp. 1129–1164.

[48] Furniss, M., *Art in motion: animation aesthetics*, John Libbey and Company ltd., Sydney, 1998.

[49] Gao, P. and Sederberg, T. W., “A work minimization approach to image morphing,” *The Visual Computer*, Vol. 14, No. 8/9, 1998, pp. 390–400.

[50] Garey, M. R. and Johnson, D. S., *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W. H. Freeman, New York, 1979.

[51] Garey, M. R. and Johnson, D. S., “Crossing number is NP-complete,” *SIAM Journal on Algebraic and Discrete Methods*, Vol. 4, No. 3, 1983, pp. 312–316.

[52] Goldstein, E. B., *Sensation and Perception*, Brooks/Cole Publishing, 5th ed., 1999.

[53] Gould, P. and White, R., *Mental Maps*, Allen and Unwin Inc., Winchester, Mass. 01890, USA, 1986.

[54] Hampel, F., Ronchetti, E., Rousseeuw, P., and Stahel, W., *Robust Statistics: The Approach Based on Influence Functions*, John Willey, NY USA, 1986.

[55] Hansen, S., Schrimpsher, D., and Narayanan, N., “Learning algorithms by visualization: A novel approach using animation-embedded hypermedia,” *In Proc. Third International Conference on The Learning Sciences, Charlottesville, VA: AACE*, 1998, pp. 125–130.

[56] Huang, M. L. and Eades, P., “A fully animated interactive system for clustering and navigating huge graphs,” *Proc. of the 6th Internat. Symposium on Graph Drawing (GD’98),* edited by S. H. Whitesides, 1998, pp. 374–383.

[57] Hutter, M. J., “Computer-generated animation finds its place in court,” The Business Review `http://albany.bcentral.com/albany/stories/1998/04/27/focus3.html`, April 27, 1998.

[58] Jünger, M. and Mutzel, P., “2-Layer straightline crossing minimization: Performance of exact and heuristic algorithms,” *Journal of Graph Algorithms and Applications*, Vol. 1, No. 1, 1997, pp. 1–25.

[59] Kamada, T. and Kawai, S., “An algorithm for drawing general undirected graphs,” *Information Processing Letters*, Vol. 31, 1989, pp. 7–15.

[60] Kaufman, L. and Rousseeuw, P. J., *Finding Groups in Data: An Introduction to Cluster Analysis*, John Wiley & Sons, NY, USA, 1990.

[61] Kaufmann, M. and Wagner, D., *Drawing graphs: methods and models*, Springer, 2001.

[62] Kehoe, C., Stasko, J., and Taylor, A., "Rethinking the evaluation of algorithm animations as learning aids: An observational study," *International Journal of Human-Computer Studies*, Vol. 54, No. 2, February 2001, pp. 265–284.

[63] Kerameos, A., Rasmussen, H., English, A., O'Sullivan, I., and Sharp, D., "Animation," `http://www.bfi.org.uk/nationallibrary/collections/alevel/animation.pdf`, 2000 bif National Library, ISBN 0 85170 825 0, note.

[64] Lawrence, A., Badre, A., and Stasko, J. T., "Empirically evaluating the use of animations to teach algorithms," *Proceedings of the 1994 IEEE Symposium on Visual Languages, St. Louis, MO*, 1994, pp. 48–54.

[65] Laybourne, K., *The animation book*, Three Rivers Press, New York, 1998.

[66] Lazarus, F. and Verroust, A., "3D metamorphosis: a survey," .

[67] Lee, S.-Y., Chwa, K.-Y., and Shin, S. Y., "Image metamorphosis using snakes and free-form deformations," *Computer Graphics*, Vol. 29, No. Annual Conference Series, 1995, pp. 439–448.

[68] Lin, X., *Analysis of Algorithms for Drawing Graphs*, Dissertation, University of Queensland, 1992.

[69] MacQueen, J., "Some methods for classification and analysis of multivariate observations," *5th Berkley Symposium on Mathematical Statistics and Probability*, edited by L. Le Cam, and J. Neyman, 1967, pp. 281–297.

[70] Magnenat-Thalman, N. and Thalman, D., *Computer Animation: Theory and Praxis*, Springer Verlag, Tokyo, 1985.

[71] Maher, M. L., "Agents as design elements in 3D virtual worlds," Departmental Seminar at the Basser Department of Computer Science, The University of Syndey, November, 14. 2001.

[72] Manning, J., *Geometric Symmetry in Graphs*, Ph.D. thesis, Purdue University, Department of Computer Sciences, 1990.

[73] Mayer, R. and Anderson, R., “The instructive animation: Helping students build connections between words and pictures in multimedia learning,” *Journal of Educational Psychology*, Vol. 84, No. 4, 1992, pp. .

[74] Mayer, R. E. and Sims, V. K., “For whom is a picture worth a thousand words? extensions of a dual-coding theory of multimedia learning,” *Journal of Educational Psychology*, Vol. 86, 1994, pp. 389–401.

[75] Mealing, S., *The art and science of computer animation*, Intellect Books, Oxford, 1992.

[76] Miller, G. A., “The magical number seven, plus or minus two: some limits on our capacity for processing information.” *The Psychological Review*, 1956, pp. 63:81–97.

[77] Misue, K. and Sugiyama, K., “An overview of diagram based idea organizer: D-abductor,” Tech. Rep. IIAS-RR-93-3E, Fujitsu Laboratories Ltd., 140 Miyamoto, Numazu-shi, Shizuoka 410-03, Japan, 1993.

[78] National Museum of Photography, Film & Telecision, “Information sheet 5.3.36,” `http://www.nmpft.org.uk/insight/info/5.3.36.pdf`.

[79] Nesbitt, K. and Friedrich, C., “Applying gestalt principles to animated visualizations of network data,” 2001, Unpublished.

[80] original source not clear. Cited e.g. in Sifianos, G., “The definition of animation: A letter from Norman McLaren,” *Animation Journal*, Vol. 3, No. 2, 1995, pp. 62–66.

[81] Ott, L. and Longnecker, M., *An introduction to statistical methods and data analysis*, Duxbury, 5th ed., 2001.

[82] P. S. Bradley, U. F. and Reina, C., “Scaling clustering algorithms to large databases,” *Proceedings of the Fourth International Conference on Knowledge Discovery and Data Mining*, edited by R. Agrawal and P. Stolorz, 1998, pp. 9–15.

[83] Pane, J. F., Corbett, A. T., and John, B. E., “Assessing dynamics in computer-based instruction,” *CHI*, 1996, pp. 197–204.

[84] Pattison, T., Vernik, R., Goodburn, D., and Phillips, M., “Rapid assembly and deployment of domain visualisation solutions.” *In proceedings of the Australian Symposium on Information Visualisation 2001*, 2001, pp. 19–26.

[85] Purchase, H. C., “Which aesthetic has the greatest effect on human understanding?” *LNCS*, edited by G. D. Battista, Vol. 1353, Springer, 1998, pp. 248–261.

[86] Purchase, H. C., “Effective information visualisation: a study of graph drawing aesthetics and algorithms,” *Interacting with Computers*, Vol. 13(2), 2000, pp. 477–506.

[87] Purchase, H. C., “Top five criteria for good graph drawings,” personal communication, 2001.

[88] Reeves, W. T., “Particle systems - a technique for modeling a class of fuzzy objects,” *ACM Transactions on Graphics*, Vol. 2, No. 2, 1983, pp. 91–108.

[89] Reynolds, C. W., “Flocks, herds, and schools: A distributed behavioral model,” *Computer Graphics*, Vol. 21, No. 4, 1987, pp. 25–34.

[90] Robertson, G. G., Card, S. K., and Mackinlay, J. D., “Information visualization using 3D interactive animation,” *Communications of the ACM*, Vol. 36, No. 4, 1993, pp. 56–71.

[91] Rousseeuw, P. and Leroy, A., *Robust regression and outlier detection*, John Willey, NY USA, 1987.

[92] Ruprecht, D. and Müller, H., “Image warping with scattered data interpolation methods,” Tech. Rep. 443, FB Informatik LS VII, University of Dortmund, 1992.

[93] Sarkar, M. and Brown, M. H., “Graphical fisheye views,” *Communications of the ACM*, Vol. 37, No. 12, 1994, pp. 73–84.

[94] Sarkar, M., Snibbe, S. S., Tversky, O. J., and Reiss, S. P., “Stretching the rubber sheet: A metaphor for viewing large layouts on small screens,” *ACM Symposium on User Interface Software and Technology*, 1993, pp. 81–91.

[95] Schroeder, B., Martin, K., and Lorensen, B., *The Visualization Toolkit: An Object-Oriented Approach to 3D Graphics*, Prentice Hall, 1996.

[96] Sederberg, T., Gao, P., Wang, G., and Mu, H., “Shape Blending: An intrinsic solution to the vertex path problem,” *In Proceedings of SIGGRAPH ’93*, Vol. 27, August 1993, pp. 15–18.

[97] Sederberg, T. W. and Greenwood, E., “A physically based approach to 2-D shape blending,” *Computer Graphics (SIGGRAPH ’92 Proceedings)*, Vol. 26, No. 2, 1992, pp. 25–34.

[98] Sekuler, R. and Blake, R., *Perception*, McGraw-Hill Publishing company, 3rd ed., 1994.

[99] Shapira, M. and Rappoport, A., “Shape blending using the Star-Skeleton representation,” *IEEE Computer Graphics and Applications*, Vol. 15, No. 2, 1995, pp. 44–50.

[100] Shepard, R. N., “Ecological constraints on internal representation: Resonant kinematics of perceiving, imagining thinking, and dreaming,” *Psychological Review*, Vol. 91, No. 4, 1984, pp. 417–447.

[101] Shoemake, K., “Animating rotation with quaternion curves,” *Computer Graphics (SIGGRAPH ’85 Proceedings)*, edited by B. A. Barsky, Vol. 19, 1985, pp. 245–254.

[102] Shoemake, K. and Duff, T., “Matrix animation and polar decomposition,” *Proc. of the 1992 Graphics Interface Conference*, 1992, pp. 245–254.

[103] Sicherman, S., “The Mud FAQ,” 2000.

[104] Small, E. S. and Levinson, E., “Toward a theory of animation,” *The Velvet Light Trap*, Vol. 24, 1989, pp. 73.

[105] Solomon, C., “Toward a definition of animation,” *The Art of Animation*, Vol. 10, 1988, pp. 9–12.

[106] Stasko, J., “SAMBA animation designer’s package,” `ftp://ftp.cc.gatech.edu/pub/gvu/tech-reports/96-19.ps.Z`, 1996.

[107] Stasko, J., Badre, A., and Lewis, C., “Do Algorithm Animations Assist Learning? An Empirical Study and Analysis,” *Proceedings of the INTERCHI ’93 Conference on Human Factors in Computing Systems*, Amsterdam, Netherlands, April 1993, pp. 61–66.

[108] Stasko, J. T., “Animating algorithms with XTANGO,” *SIGACT News*, Vol. 23, No. 2, Spring 1992, pp. 67–71.

[109] Stollnitz, E. J., DeRose, T. D., and Salesin, D. H., “Wavelets for computer graphics: A primer, part 1,” *IEEE Computer Graphics and Applications*, Vol. 15, No. 3, 1995, pp. 76–84.

[110] Sugiyama, K., “A cognitive approach for graph drawing,” *Cybernetics and Systems: An International Journal*, Vol. 18, 1987, pp. 447–488.

[111] Sugiyama, K., Tagawa, S., and Toda, M., “Methods for visual understanding of hierachical system structures,” *IEEE Transactions of Systems, Man and Cybernetics*, Vol. 11, 1981, pp. 109–125.

[112] Sugiyama, K. and Toda, M., “Structuring information for understanding complex systems,” *IEEE Trans. on Systems, Man and Cybernetics*, Vol. SMC-11, No. 2, 1985, pp. 109–125.

[113] Tolman, E., “Cognitive maps in rats and men,” *Psychological Review*, Vol. 55, 1948, pp. 189–208.

[114] Tu, X. and Terzopoulos, D., “Artificial fishes: Physics, locomotion, perception, behavior,” *Computer Graphics*, Vol. 28, No. Annual Conference Series, 1994, pp. 43–50.

[115] Tufte, E. R., *The Visual Display of Quantitative Information*, Graphics Press, 1986.

[116] Tufte, E. R., *Envisioning Information*, Graphics Press, 1991.

[117] Tufte, E. R., *Visual Explanations*, Graphics Press, 1998.

[118] U. Fayyad, C. R. and Bradley, P. S., “Initialization of iterative refinement clustering algorithms,” *Proceedings of the Fourth International Conference on Knowledge Discovery and Data Mining*, edited by R. Agrawal and P. Stolorz, 1998, pp. 194–198.

[119] `http://www.mpi-sb.mpg.de/AGD/`, *The AGD-Library User Manual Version 1.1.2*, Max-Planck-Institut für Informatik.

[120] van Dam S. K. Feiner, J. D. F. A. and Hughes, J. F., *Computer graphics, principles and practice*, Addison-Wesley, 1997.

[121] Walsh, S., “Saudi told to pay $1 billion in BCCI case, testimony turns to levity in court,” Washington Post, 6/25/1999.

[122] Watt, A. and Watt, M., *Advanced Animation and Rendering Techniques*, Addison Wesley, 1992.

[123] Webber, R. J., *Finding the Best Viewpoints for three-dimensional graph drawings*, Ph.D. thesis, University of Newcastle (Australia), 1998.

[124] Whitesides, S., “Chain reconfiguration,” *ISAAC 2001*, edited by P. Eades and T. Takaoka, LNCS 2223, Springer Verlag, 2001, pp. 1–13.

[125] Wolberg, G., *Digital Image Warping*, IEEE Computer Society Press, Los Alamitos, CA, 1990.

[126] Wolberg, G., "Recent advances in image morphing," *In Proc. Computer Graphics Internat. 1996*, 1996, pp. 64–71.

[127] Wolberg, G., "Image morphing: a survey," *The Visual Computer*, Vol. 14, No. 8/9, 1998, pp. 360–372.

[128] Yee, K.-P., Fisher, D., Dhamija, R., and Hearst, M., "Animated exploration of dynamic graphs with radial layout," *Proceedings of Information Visualization 2001*, San Diego, CA, October 2001, pp. 43–50.

[129] Zevitas, C., "Fear of death nets $2.2 million for 13 airline passengers with only minor physical injuries," Lawyers Weekly USA, 11/29/1999.

[130] Zhang, Y. and Huang, Y., "Wavelet shape blending," *The Visual Computer*, Vol. 16, No. 2, 2000, pp. 106–115.

APPENDIX A

Data Tables

Method	Scenario	SEC	UNM	CEL	TEC	NPL
Linear	1	130	5.020E+09	1,843,276	2,095	347,983
Ortho	1	103	3.589E+09	3,418,826	1,990	439,772
Force	1	40	9.840E+08	3,312,649	701	423,831
Spring 3D	1	119	2.376E+09	133,613	1,229	448,822
Regression	1	65	1.722E+09	1,203,360	490	404,066
Cluster	1	388	4.538E+09	41,883,085	3,765	445,023
Best	1	40	9.840E+08	133,613	490	347,983
Median	1	111	2.983E+09	2,577,963	1,610	431,802
Linear	1b	130	5.020E+09	1,843,276	2,138	347,983
Ortho	1b	145	5.206E+09	2,519,567	3,345	439,772
Force	1b	64	1.574E+09	2,450,485	635	524,618
Spring 3D	1b	117	2.278E+09	130,946	1,235	449,260
Regression	1b	65	1.722E+09	1,203,360	533	404,055
Cluster	1b	335	6.340E+09	28,847,622	4,254	491,297
Best	1b	64	1.574E+09	130,946	533	347,983
Median	1b	124	3.649E+09	2,146,880	1,687	444,516
Linear	2	12,891	1.675E+10	212,992	1,301,941	440,110
Ortho	2	6,811	5.489E+09	106,560	687,955	532,660
Force	2	179	8.419E+08	27,090	6,358	634,338
Spring 3D	2	4	9.863E+06	125	413	950,719
Regression	2	6	1.066E-20	0	578	691,249
Cluster	2	6	1.234E-20	0	560	691,294
Best	2	4	1.066E-20	0	413	440,110
Median	2	92	4.259E+08	13,608	3,468	662,794
Linear	2b	9,681	1.670E+10	202,046	582,018	439,723
Ortho	2b	1,695	5.484E+09	103,893	72,066	532,091
Force	2b	518	8.581E+08	37,058	11,203	635,514
Spring 3D	2b	158	1.084E+07	364	793	949,914
Regression	2b	105	6.805E+03	895	612	690,308
Cluster	2b	176	4.184E+06	11,733	1,398	690,412
Best	2b	105	6.805E+03	364	612	439,723
Median	2b	347	4.345E+08	24,396	6,301	662,911
Linear	3	40	3.962E+08	114,892	4,055	94,772
Ortho	3	24	3.194E+08	52,429	2,428	125,880
Force	3	1	1.477E+08	23,421	56	130,323
Spring 3D	3	16	1.569E+08	1,845	260	167,423
Regression	3	0	2.121E+08	48,123	0	113,145
Cluster	3	0	1.039E+08	720	0	132,007
Best	3	0	1.039E+08	720	0	94,772
Median	3	8	1.845E+08	35,772	158	128,102

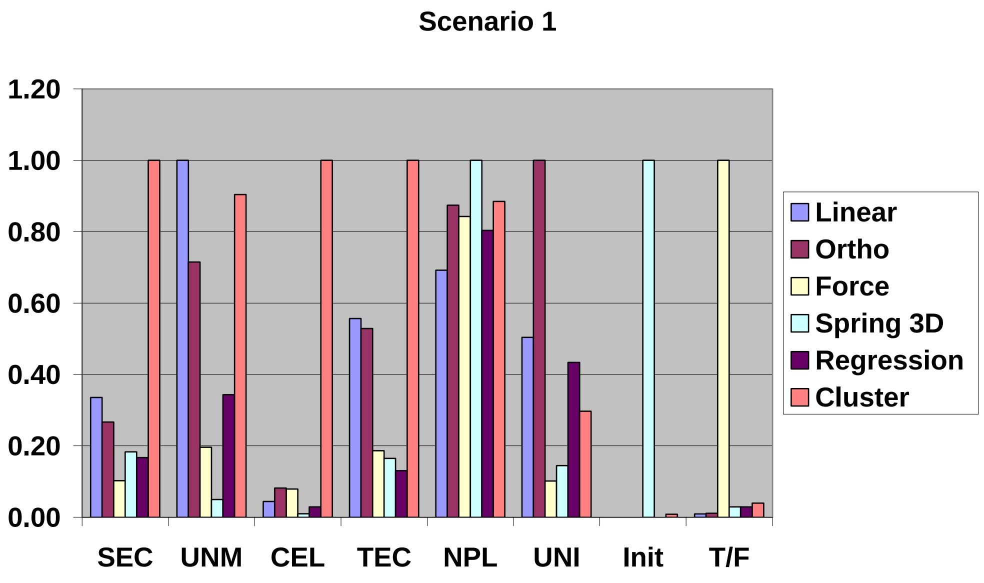
Method	Scenario	UNI	Frames	Init	Frame time	T/F
Linear	1	129	101	0	50	0.5
Ortho	1	256	101	0	60	0.6
Force	1	26	101	0	5560	55.0
Spring 3D	1	52	101	68,160	160	1.6
Regression	1	111	101	0	160	1.6
Cluster	1	76	101	270	220	2.2
Best	1	26	101	0	50	0
Median	1	94	101	0	160	1.6
Linear	1b	129	101	0	0	0.0
Ortho	1b	283	101	0	145	1.4
Force	1b	19	101	0	5270	52.2
Spring 3D	1b	55	101	44,930	60	0.6
Regression	1b	111	101	0	890	8.8
Cluster	1b	104	101	380	1200	11.9
Best	1b	19	101	0	0	0
Median	1b	108	101	0	518	5.1
Linear	2	3,813,032	101	50	160	1.6
Ortho	2	347,943	101	50	160	1.6
Force	2	49,158	101	0	297250	2,943.1
Spring 3D	2	0	101	12,960	110	1.1
Regression	2	0	101	490	780	7.7
Cluster	2	0	101	2,200	1920	19.0
Best	2	0	101	0	110	1
Median	2	24,579	101	270	470	4.7
Linear	2b	3,577,834	101	0	100	1.0
Ortho	2b	322,941	101	0	210	2.1
Force	2b	47,201	101	0	368440	3,647.9
Spring 3D	2b	5,177	101	122,650	2370	23.5
Regression	2b	954	101	60	590	5.8
Cluster	2b	20,013	101	3,460	1800	17.8
Best	2b	954	101	0	100	1
Median	2b	33,607	101	30	1,195	11.8
Linear	3	23,519	101	0	60	0.6
Ortho	3	2,916	101	0	0	0.0
Force	3	335	101	0	2920	28.9
Spring 3D	3	621	101	120,290	60	0.6
Regression	3	19	101	0	170	1.7
Cluster	3	0	101	110	340	3.4
Best	3	0	101	0	0	0
Median	3	478	101	0	115	1.1

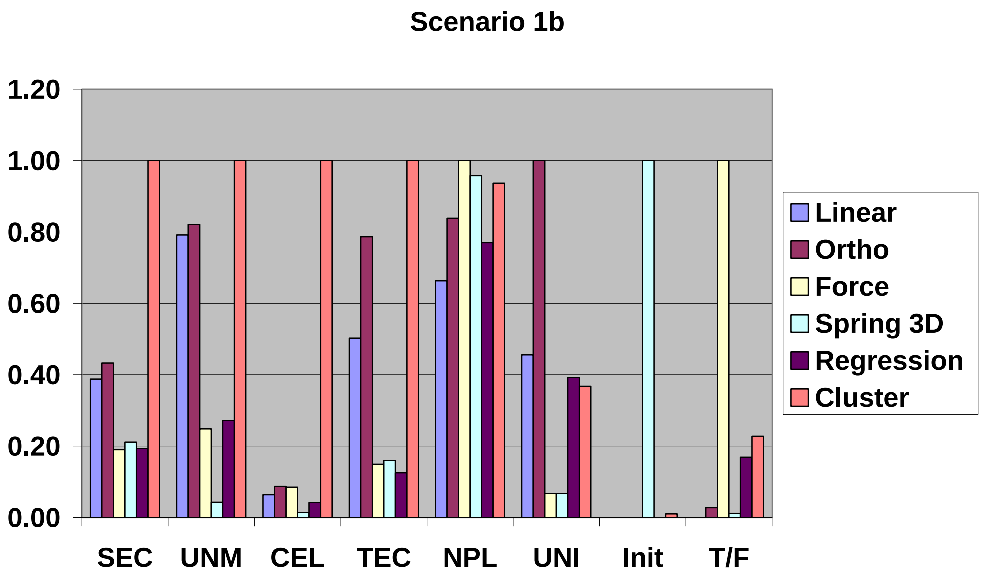
Method	Scenario	SEC	UNM	CEL	TEC	NPL
Linear	4	190	2.877E+08	1,326,461	1,328	120,561
Ortho	4	389	6.645E+08	3,446,831	3,509	155,495
Force	4	258	3.104E+08	4,685,660	985	141,709
Spring 3D	4	225	3.124E+08	136,038	1,435	174,306
Regression	4	164	4.306E+08	2,341,735	1,066	130,556
Cluster	4	197	2.441E+08	1,909,777	1,160	127,250
Best	4	164	2.441E+08	136,038	985	120,561
Median	4	211	3.114E+08	2,125,756	1,244	136,133
Linear	5	420	1.470E+10	2,399,401	15,725	597,271
Ortho	5	1,174	1.245E+10	3,309,387	22,711	761,707
Force	5	327	5.029E+09	3,214,709	15,021	700,593
Spring 3D	5	895	1.183E+10	1,638,216	21,350	868,716
Regression	5	308	1.479E+10	2,626,936	13,095	763,763
Cluster	5	1,081	1.151E+10	43,154,023	8,438	673,484
Best	5	308	5.029E+09	1,638,216	8,438	597,271
Median	5	658	1.214E+10	2,920,822	15,373	731,150
Linear	6	7,095	2.385E+08	4,174,533	15,762	73,957
Ortho	6	6,788	2.044E+08	4,528,321	19,494	94,808
Force	6	4,853	2.111E+08	9,190,151	15,458	84,447
Spring 3D	6	8,200	2.143E+08	532,111	14,227	162,877
Regression	6	6,725	4.998E+08	10,812,041	8,301	118,195
Cluster	6	8,040	2.009E+08	9,184,635	16,768	88,536
Best	6	4,853	2.009E+08	532,111	8,301	73,957
Median	6	6,942	2.127E+08	6,856,478	15,610	91,672
Linear	7	1,011	4.858E+07	90,948	4,975	25,170
Ortho	7	873	4.028E+07	953,805	4,942	28,418
Force	7	518	1.841E+07	956,999	2,905	40,229
Spring 3D	7	697	3.280E+07	44,187	8,500	61,502
Regression	7	454	4.496E+07	678,300	3,718	59,502
Cluster	7	1,195	9.394E+07	11,209,526	4,916	50,338
Best	7	454	1.841E+07	44,187	2,905	25,170
Median	7	785	4.262E+07	816,052	4,929	45,283
Linear	8	46,185	2.829E+09	4,138	2,112,739	290,510
Ortho	8	40,254	7.195E+08	22,157	2,002,272	372,986
Force	8	28,706	8.743E+08	42,538	1,964,892	380,518
Spring 3D	8	44,363	1.005E+09	6,499	2,040,526	1,081,968
Regression	8	36,894	1.007E+09	7,932	1,962,955	326,451
Cluster	8	45,209	2.879E+09	453,584	289,169	341,731
Best	8	28,706	7.195E+08	4,138	289,169	290,510
Median	8	42,309	1.006E+09	15,045	1,983,582	357,359

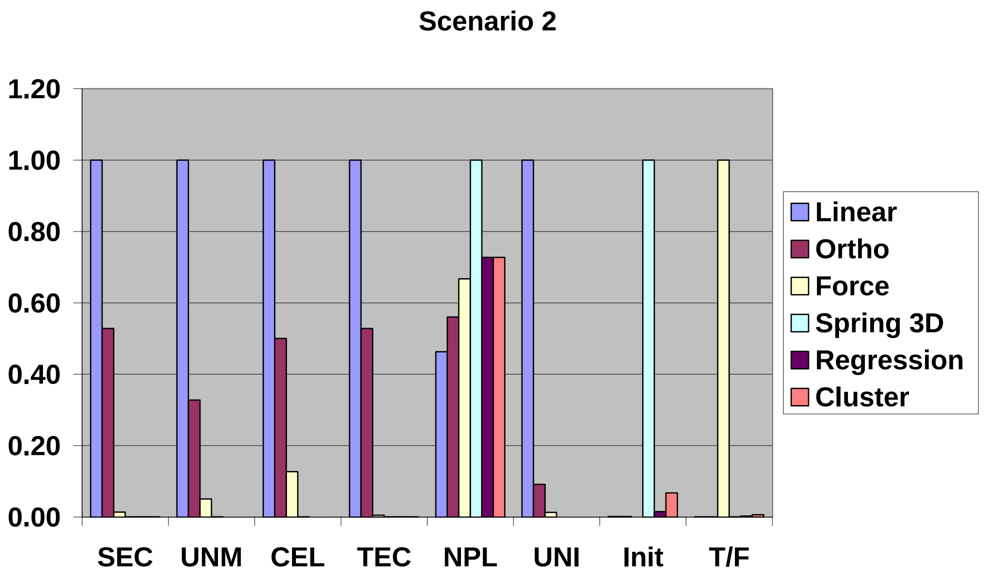
Method	Scenario	UNI	Frames	Init	Frame time	T/F
Linear	4	169	101	0	0	0.0
Ortho	4	269	101	0	50	0.5
Force	4	56	101	0	1140	11.3
Spring 3D	4	195	101	9,010	50	0.5
Regression	4	150	101	0	0	0.0
Cluster	4	97	101	220	60	0.6
Best	4	56	101	0	0	0
Median	4	160	101	0	50	0.5
Linear	5	277	101	0	0	0.0
Ortho	5	911	101	0	0	0.0
Force	5	48	101	0	11680	115.6
Spring 3D	5	195	101	2,310	120	1.2
Regression	5	307	101	110	170	1.7
Cluster	5	266	101	550	290	2.9
Best	5	48	101	0	0	0
Median	5	272	101	55	145	1.4
Linear	6	185	101	0	50	0.5
Ortho	6	144	101	0	0	0.0
Force	6	44	101	0	840	8.3
Spring 3D	6	89	101	12,900	0	0.0
Regression	6	288	101	50	0	0.0
Cluster	6	192	101	330	50	0.5
Best	6	44	101	0	0	0
Median	6	165	101	25	25	0.2
Linear	7	2,692	101	0	0	0.0
Ortho	7	1,305	101	0	0	0.0
Force	7	77	101	0	390	3.9
Spring 3D	7	483	101	12,970	100	1.0
Regression	7	198	101	50	0	0.0
Cluster	7	197	101	220	230	2.3
Best	7	77	101	0	0	0
Median	7	341	101	25	50	0.5
Linear	8	7,427,591	101	50	730	7.2
Ortho	8	1,470,055	101	50	420	4.2
Force	8	1,385,040	101	0	1385040	13,713.3
Spring 3D	8	2,369,383	101	87,500	430	4.3
Regression	8	2,915,604	101	110	2030	20.1
Cluster	8	3,507,042	101	5,660	3300	32.7
Best	8	1,385,040	101	0	420	4
Median	8	2,642,494	101	80	1,380	13.7

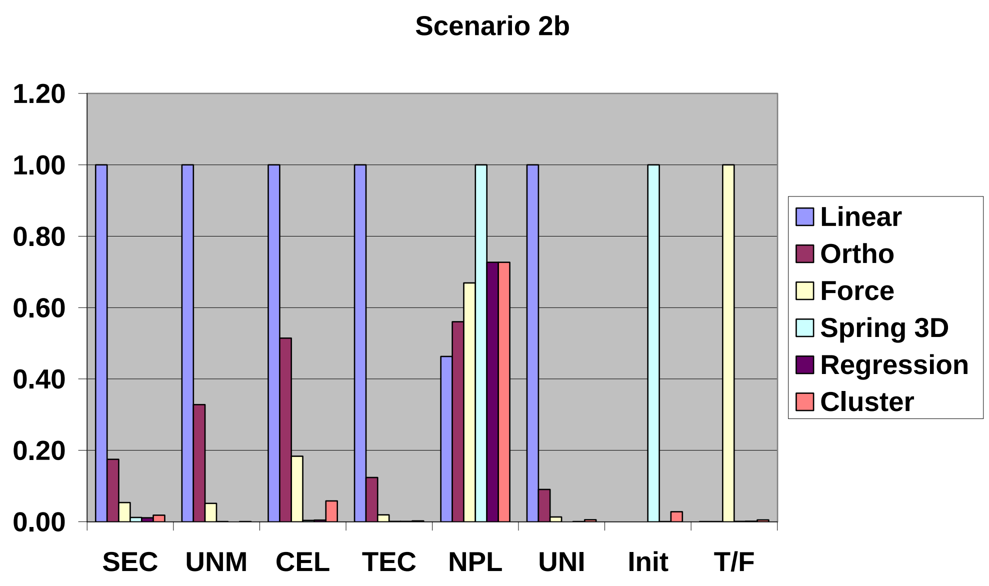
APPENDIX B

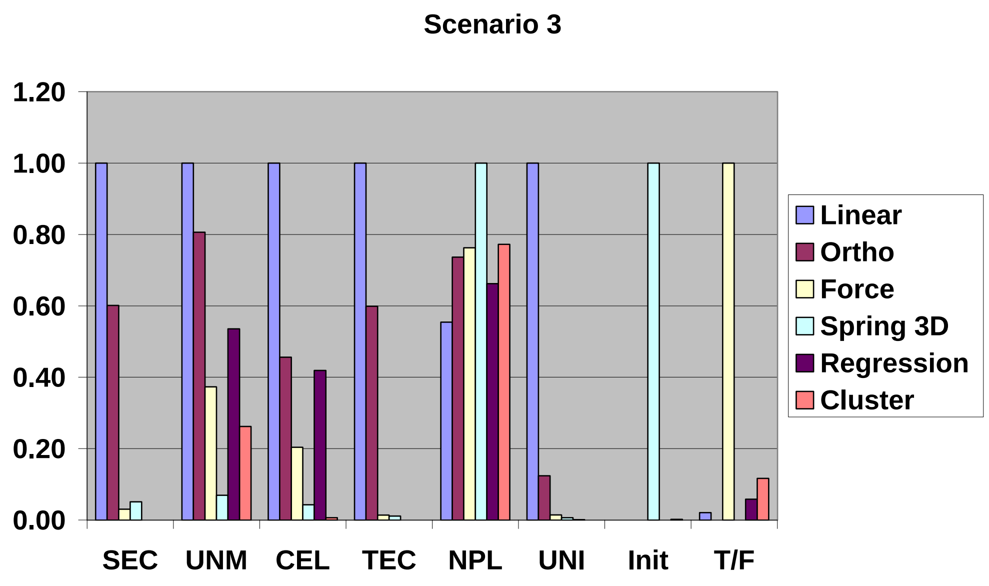
Graphical Presentation of Benchmark Results

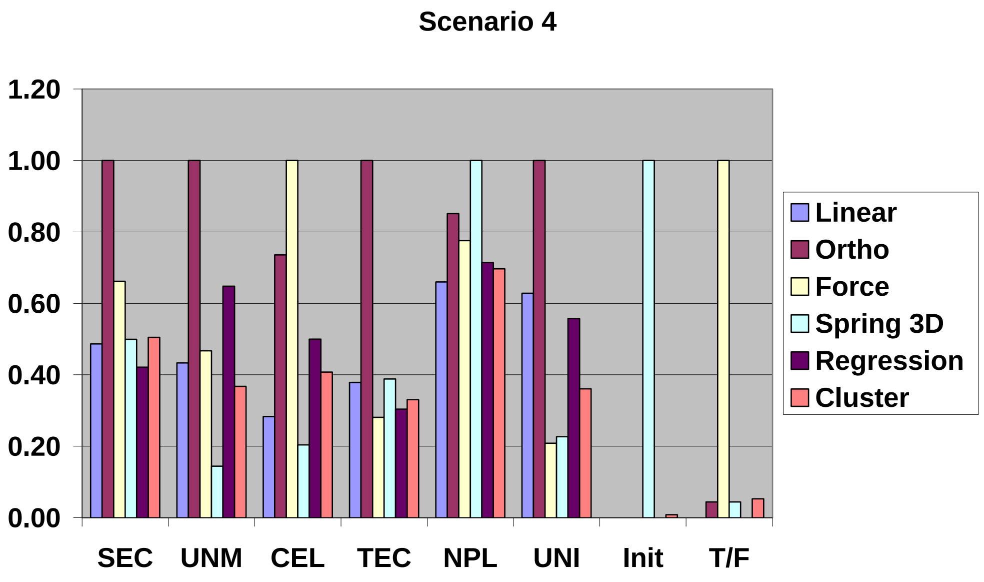


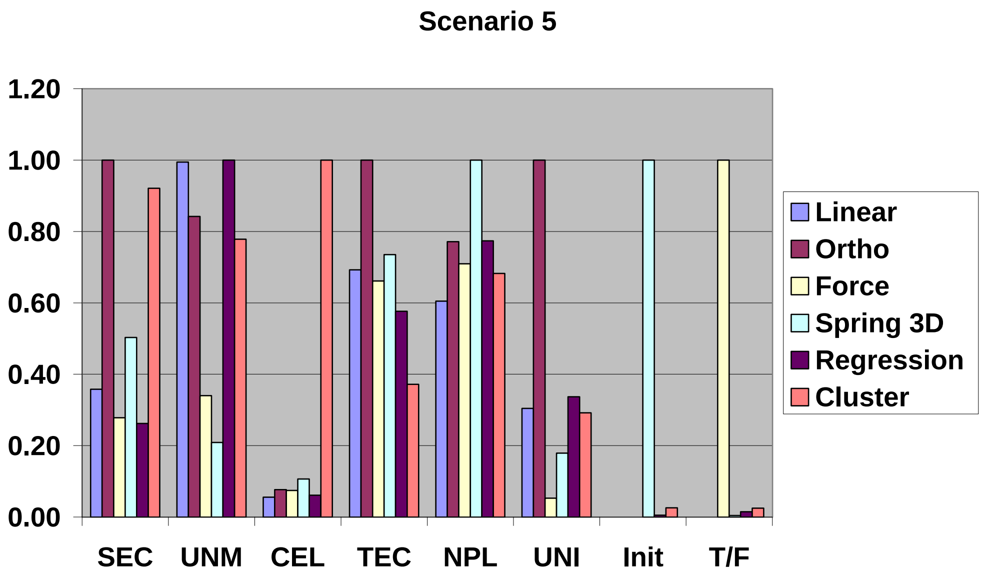

Scenario 1b
1.20
1.00
0.80
0.60
0.40
0.20
0.00
SEC
UNM
CEL
TEC
NPL
UNI
Init
T/F
Linear
Ortho
Force
Spring 3D
Regression
Cluster

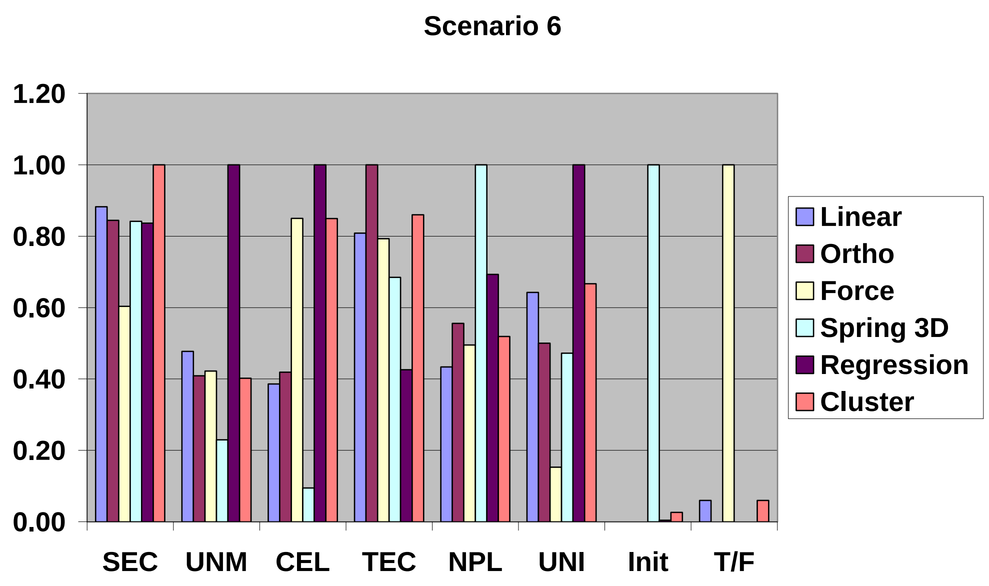

Scenario 2
1.20
1.00
0.80
0.60
0.40
0.20
0.00
SEC
UNM
CEL
TEC
NPL
UNI
Init
T/F
Linear
Ortho
Force
Spring 3D
Regression
Cluster

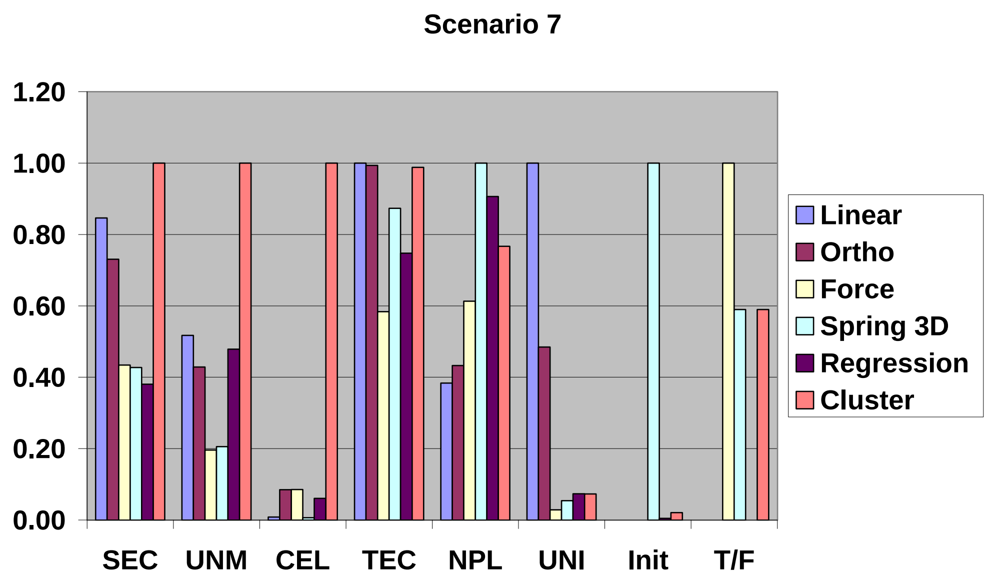

Scenario 2b
1.20
1.00
0.80
0.60
0.40
0.20
0.00
SEC
UNM
CEL
TEC
NPL
UNI
Init
T/F
Linear
Ortho
Force
Spring 3D
Regression
Cluster

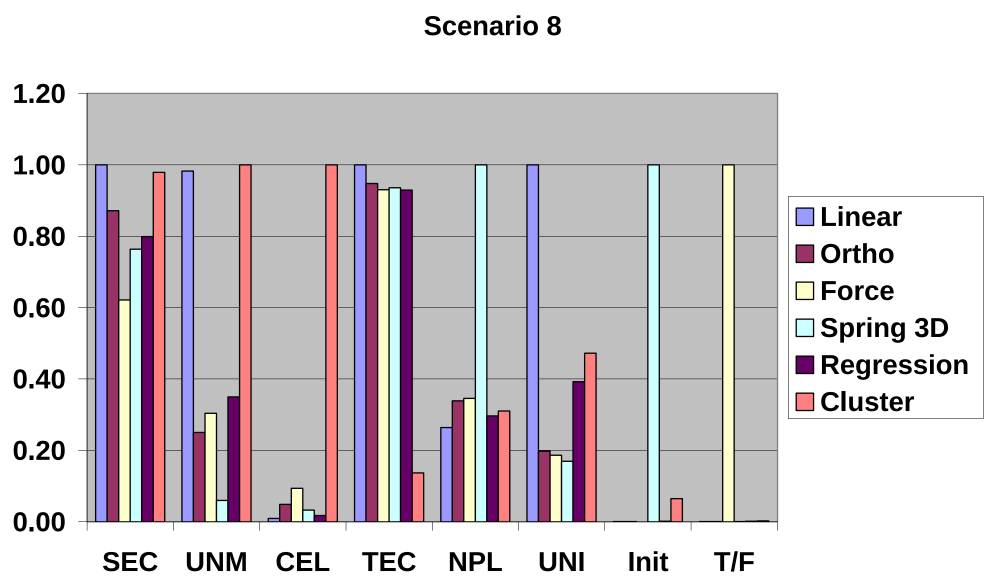

Scenario 3
1.20
1.00
0.80
0.60
0.40
0.20
0.00
SEC
UNM
CEL
TEC
NPL
UNI
Init
T/F
Linear
Ortho
Force
Spring 3D
Regression
Cluster


Scenario 4
1.20
1.00
0.80
0.60
0.40
0.20
0.00
SEC
UNM
CEL
TEC
NPL
UNI
Init
T/F
Linear
Ortho
Force
Spring 3D
Regression
Cluster


Scenario 5
1.20
1.00
0.80
0.60
0.40
0.20
0.00
SEC
UNM
CEL
TEC
NPL
UNI
Init
T/F
Linear
Ortho
Force
Spring 3D
Regression
Cluster


Scenario 6
1.20
1.00
0.80
0.60
0.40
0.20
0.00
SEC
UNM
CEL
TEC
NPL
UNI
Init
T/F
Linear
Ortho
Force
Spring 3D
Regression
Cluster


Scenario 7
1.20
1.00
0.80
0.60
0.40
0.20
0.00
SEC
UNM
CEL
TEC
NPL
UNI
Init
T/F
Linear
Ortho
Force
Spring 3D
Regression
Cluster


Scenario 8
1.20
1.00
0.80
0.60
0.40
0.20
0.00
SEC
UNM
CEL
TEC
NPL
UNI
Init
T/F
Linear
Ortho
Force
Spring 3D
Regression
Cluster