University of Bergen
Department of Informatics

TimeBender: Interactive Authoring of 3D Space-Time Narratives

Author: Rikke Aas
Supervisor: Stefan Bruckner

UNIVERSITETET I BERGEN
Det matematisk-naturvitenskapelige fakultet

September, 2022

Abstract

Communication of scientific results and discoveries to, for example, fellow domain experts, business partners, students, or the general public, is an important part of research. Communication through visualization has been proven to be effective when the representations are memorable and engaging, and research has shown that these communicative visualizations can be further enhanced with narratives for certain audiences. A challenge faced by scientists is to create memorable and engaging visualizations for communication which traditionally has been done by trained illustrators and designers.

Therefore, we created TimeBender, a framework and prototype implementation to bridge this gap specifically for authoring narrative posters in a 3D environment with a space-time dimension. The posters feature multiple scenes forming the narrative, which are connected by an elongated object encoding the narrative flow. We demonstrate that our approach is capable of aiding the authoring of these posters through a 3-step pipeline where, first, the scenes are set up individually, then, the global layout of scenes in the poster space is determined, before details, such as textual elements, are added. TimeBender supports animation as each scene is rendered dynamically within the poster. The framework and example results were evaluated in an expert interview with a professional illustrator.

Contents

1 Introduction **1**

1.1 Problem Statement and Contribution 1

2 Related Work **5**

2.1 Visual Communication 5

2.2 Illustrative Visualization 10

2.2.1 Low-Level Visual Abstractions 11

2.2.2 High-Level Visual Abstractions 14

2.3 Narrative Visualization 17

3 Methodology **23**

3.1 Main Pipeline 24

3.1.1 Local Layout 24

3.1.2 Global Layout 25

3.1.3 Detailed Layout 29

3.2 Focus Object Approaches 32

3.2.1 Viewpoint-Based Approach 33

3.2.2 Spline-Based Approach 36

3.3 Soft Fading Effects 37

4 Implementation **42**

4.1 Unreal Engine 42

4.1.1 Project Overview 43

4.2 Main Pipeline 45

4.2.1 Local Layout 45

4.2.2 Global Layout 47

4.2.3 Detailed Layout 50

4.3 Focus Object Modelling 52

4.4 Soft Fading Effects 53

4.5 Saving and Loading 56

5 Results **59**

5.1 Performance 59

5.2 Usage Examples 60

5.2.1 The Genetic Journey 60

5.2.2 7 Wonders in 13 Days 61

5.3 Expert Feedback 61

6 Discussion and Limitations **68**

7 Conclusion and Future Work **71**

List of Acronyms and Abbreviations **73**

Bibliography **74**

Chapter 1

Introduction

When the Chauvet cave was discovered in France in 1994, it provided tangible proof that our ancestors used images to communicate more than 30,000 years ago. An example from these cave paintings is shown in Figure 1.1. To date, visualizations are used to enhance communication of, for example, scientific results, news stories, or instructions on how to build a Lego. Although it has a long history of use and wide area of applicability, there is still much research needed in the field of visualization for communication. Research that has been conducted shows that communicative visualizations are different from analytic and exploratory visualizations in that they should aim at being memorable and engaging [49]. What makes a visualization memorable and engaging is a highly discussed topic in the scientific community. There is, however, evidence from published studies [6, 9] pointing towards making embellished and unique visualizations to increase memorability, and using storytelling to engage the audience [29, 55, 56]. In this thesis, we therefore create communicative visualizations that combine visual interest through traditional illustration techniques, with visual storytelling for efficient communication.

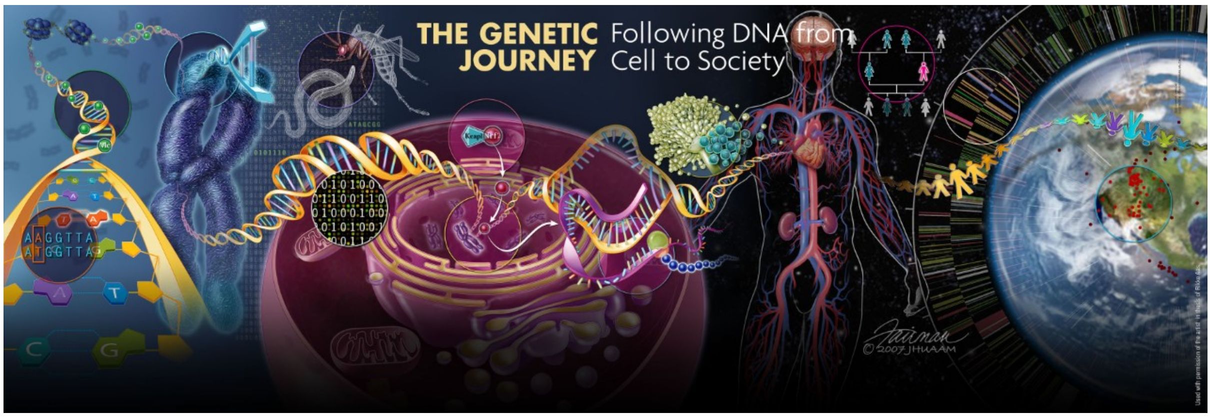
1.1 Problem Statement and Contribution

Presentation is especially important for outreach and dissipation of scientific results to the public. Telling a story with the findings and using engaging visuals to support this will keep the attention and interest of the audience, and can make complex scientific topics easier to grasp. However, creating such narrative visualisations for communication manually is a time consuming task which often requires artistic skills. Hiring an artist

Figure 1.1: A famous section of the paintings in the Chauvet cave depicting horses [72]

or graphical designer to do this job is an option, but the problem then becomes to accurately describe what the content of the visualization should be. The artist will probably not be an expert in the field and thus there is a large potential for misunderstandings, details being lost, or focus being shifted. It is desirable for domain experts to create the visualizations themselves, but without the skills of an artist or designer, they need tools to help them.

This is a broad problem, and creating a general solution for all types of visual communication of scientific results is not feasible. According to Kosara [49], visual communication techniques should be specific and not general, and should aim at being highly useful for particular data sets and use cases rather than being generally useful. Therefore, we chose to focus our work on poster visualizations with a certain format. This format includes multiple scenes that form a narrative in addition to an elongated object, which we call the focus object, that ties all the scenes together and represents the narrative flow. Each scene interacts with, or explains a certain part of, the focus object. A motivating example from the traditional illustrator Jennifer Fairman can be seen in Figure 1.2. Here, the DNA strand is the elongated focus object that ties the scenes together, roughly from left to right. The poster has similarities to comics-based visualizations, where different scenes layouted on a page form a narrative. The placement and size of each scene are important considerations for the author of a comics-based visualization, which is also true when using TimeBender. A crucial difference between the format of our posters and comics-based visualizations is the focus object that encodes the narrative flow. This allows the author more freedom when placing scenes in the poster, as the audience can follow the focus object along the narrative flow rather than the position of the scenes implicitly encoding this.



Credit: Illustration by Jennifer E. Fairman, CMI, FAMI © 2007 JHU AMM

Figure 1.2: The Genetic Journey by Fairman Studios [69], a motivating example for the work in this thesis. The DNA strand smoothly ties the scenes together, in addition to being part of the scenes. This image is included with the permission of Fairman Studios, all rights reserved.

The main contribution of this project is a framework, with an implemented prototype, for authoring narrative posters in a 3D environment with an additional time dimension. In contrast to comics-based visualizations, the time dimension is not implicitly encoded in the layout of scenes, but explicitly encoded with the elongated focus object. The posters can also contain animations given that they are presented digitally, which could be considered as a dual time dimension. Our framework creates posters in a 3D environment which means that the scenes are set up with 3D models in a 3D space, i.e., in addition to moving objects horizontally and vertically along the screen or poster, the author can move them in depth.

To create a cohesive poster with the described format, one major challenge was to ensure that the focus object ties the scenes together smoothly and continuously, while interacting with the scenes as defined by the poster author. We call this *inter-scene* and *intra-scene* behaviour, i.e., how the focus object behaves between scenes and how the focus object behaves within a scene respectively. To ensure well-defined inter- and intra-scene behaviour, the focus object is rendered with only one perspective camera, the main camera of the poster, while the other objects in a scene are rendered by a separate camera, the scene camera. Each scene has its own scene camera, and the rendered images from these are stitched together with the main camera image to create the complete poster. The authoring process is interactive, so the author of a poster is able to set up different scenes, place them in the poster, and influence the shape and positioning of the focus object, while the framework conserves the expected inter- and intra-scene behaviour.

The narratives in the posters created with our framework have a linear structure, i.e., the sequence of scenes follows one after another without diverging paths. This is also known as an author-driven narrative as the audience cannot influence the narrative flow by, for example, choosing different paths. While interaction by the audience is interesting as a potential future extension to TimeBender, it is not within the scope of this thesis. The narratives can also be comparative by placing scenes that should be compared close together since we allow for any layout of the scenes within a poster.

To evaluate our approach, as well as the quality of example posters created with our prototype, we conducted an expert interview with a professionally trained illustrator. Interviewing an illustrator made it possible to compare our method to the normal workflow of an illustrator, as well as verifying that our posters comply with illustration best practices.

Chapter 2

Related Work

Although communication as a main task of visualizations has not been a central focus in the visualization research community, there has been scientific work on the topic. We discuss the importance of communication as a visualization research area, as well as how to create a well-performing communicative visualization, which are also called presentation-oriented visualizations. This includes making the visualizations memorable and engaging, which can be done by using illustrative visualization to create visual interest, and narrative visualization to invoke emotions and engage the audience.

2.1 Visual Communication

The three pillars of visualization research are data, users, and tasks, hence visualization solutions usually target a specific type of data, group of users, and set of tasks. To create more generalizable visualizations, efforts have been made to abstract from domain specific tasks to pure visualization tasks. In her book, Munzner defines three levels of task abstraction: low-level querying, mid-level searching, and high-level analyzing tasks [58]. Each level is in turn split into different cases. In this task abstraction system, communication is defined as a sub-task of analyzing and given little importance. Recent work by Garrison et al. [37] uses a simpler task abstraction, compared to the one presented by Munzner, which defines three high-level tasks: exploration - to generate hypothesis, analysis - to verify hypothesis, communication - to dissipate findings. In this classification, communication is a high-level task in its own right. This could indicate a trend in the visualization research community, where visualization for communication and presentation of findings is being treated as an independent topic of research and not an afterthought in tools and frameworks built for analysis or exploration. This is also emphasized by Kosara who argues for the need to consider communication as an equally important task in visualization research as exploration and analysis [49]. He states that when presenting data, the goal is to leave a lasting impression by being *memorable* and *engaging*, which standard visualization techniques, e.g., bar charts, scatter plots, etc., generally are not. Kosara makes the point that these classic visualization techniques are not failing at these goals, they are simply not attempting to achieve them. Memorability and engagement are not important factors in analysis and exploration tasks, and can in contrast be harmful to the execution of these tasks by distracting the user. However, they are crucial for visual communication and we will take a closer look at how to achieve these goals in the following sections.

Visualizations can also be abstracted with respect to the design aspects. Moere and Purchase, inspired by Roman architecture, argue that there are three equally important design aspects in a visualization: utility, soundness, and aesthetics [73]. In traditional visualization research, the focus has been on the first two aspects where utility is related to how effective and efficient a visualization is, i.e., how well the tasks are performed and how fast these are achieved, and the soundness is related to the implementation and generalizability, e.g., to different data sets and tasks. The authors attempt to draw research focus onto the third part of the triad: aesthetics or attractiveness, by stating that more aesthetic visualizations can lead to more user engagement and thus also more efficient communication in the case of a communicative visualization.

Memorability

What makes a visualization memorable has been a highly debated topic in the visualization research community. A natural instinct could be to think that visual embellishments create memorable visualizations since they stand out. But visual embellishments have also been named "chartjunk" by Tufte [71] and his supporters, and presented as something to avoid in visual representations of data. To provide evidence for the link between memorability and embellishment, Bateman et al. performed a study where they let participants see either a minimal or embellished version of the same 14 data charts [6]. Examples of a minimal (a) and embellished (b) version of the same data chart are shown in Figure 2.1. The authors then asked half of the participants to recall the charts they had seen immediately, whereas the other half were invited back after two to three weeks to recall the charts. They found that there was no significant difference between the two

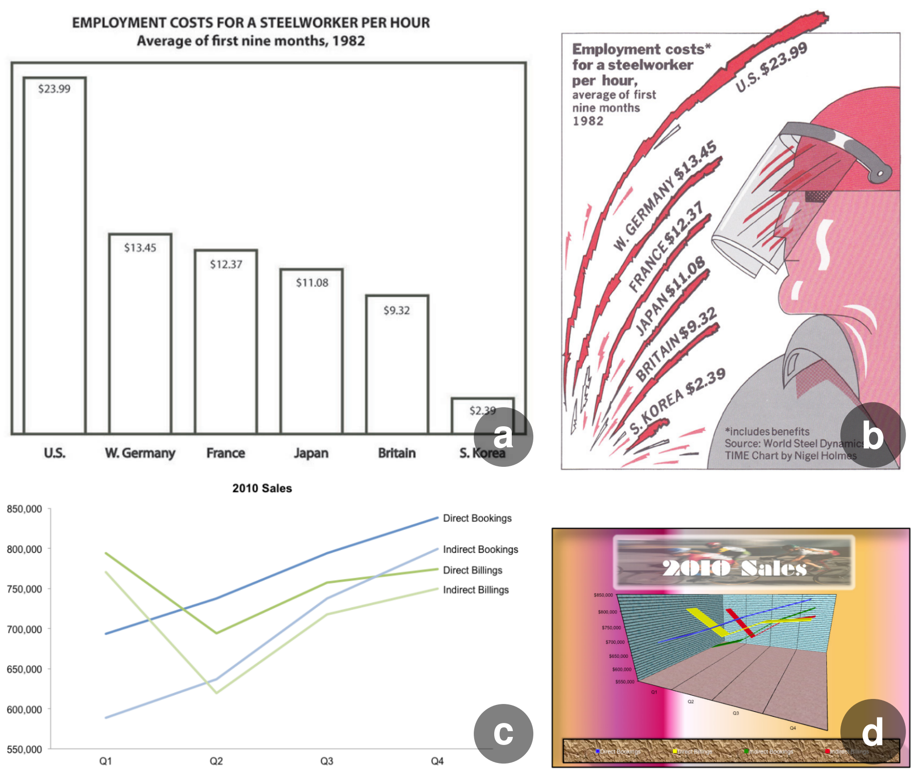


Figure 2.1: Example from the memorability study of Bateman et al. [6] with (a) a plain visualization free of ”chartjunk” and (b) an embellished visualization of the same data. In addition, there is an example from the critical response to this study by Few [32], again with (c) a plain visualization and (d) an embellished visualization of the same data. Clearly, the usefulness of embellishment can vary.

chart types with the immediate recall, whereas with the recall after several weeks, the embellished charts performed significantly better. However, the findings in this study have been questioned by, among others, Few [32]. He criticized the study design in multiple ways, e.g., the number and diversity of participants (the study was conducted on 20 university students), as well as the visual prompts in the study. According to Few, the minimal charts were designed badly creating an unfair comparison, while the embellished charts were relatively uniform in that they were all simple charts with one concise message and designed by the same established graphical designer, Nigel Holmes. Few argued that the study made claims that were much bigger than what they were actually proving, as they were only investigating a very limited set of embellished charts, not embellishment in general. Figure 2.1 c-d shows a counter example of useful embellishment, according to Few.

To overcome the criticized limitations of the study by Bateman et al., and get closer to resolving the debate, Borkin et al. performed a new and more comprehensive study on the same topic [9]. The study design was different in that they used only real-world visualizations, in addition to having a larger number of both visual prompts and participants, with 2070 and 261, respectively. A finding of the study was that more cluttered visualizations, including embellishments, were indeed more memorable than minimalistic visualizations. In addition, including color and human recognizable objects, as well as using novel visualization types, have a favorable impact on the memorability. The findings from this study imply that posters created with TimeBender may be memorable as they are rather embellished than minimalistic and feature 3D models which are closer to human recognizable objects than abstract representations of data. The results of the study by Borkin et al. seemingly contradict the views of Tufte and Few, the former famously stating in his first book: "Graphical excellence is that which gives the viewer the greatest number of ideas in the shortest time with the least ink in the smallest space" [71]. But in fact, the minimalist views of these scientists do not exclude embellished visualizations. Few affirms that embellishments can actually be useful when they support the message that the visualization is trying to communicate by drawing attention to salient features, making the message more memorable, and creating more engagement [32]. Figure 2.2 shows two visualizations featured by Tufte as examples of graphical excellence, showing that message-supporting embellishment is valued also by the founder of the "chartjunk" debate.

Embellished visualizations, such as the one shown in Figure 2.1 b, are traditionally created manually by graphical designers or illustrators. Efforts have been made in visualization research to simplify the process. For example, the work by Kim et al. [46] aimed at helping designers create embellished data charts that remain true to the underlying data by adding Data-Driven Guides, i.e., indicators of, for example, length or size that the designers could use to measure their hand-drawn components. Wang et al. [76] created a tool to allow users to customize the marks of a regular data graphic, e.g., a bar chart or line chart, to easily create embellished versions of their data graphics. Park et al. [61] focus on line charts and find photos with a similar linear structure, for example, the silhouette of a mountain range. The photo is then manipulated to fit the line chart exactly and the result is a Graphoto, an embellished chart combining the line chart with a matching photograph. These examples all focus on data that does not have an intrinsic spatial arrangement. For data with an intrinsic spatial arrangement, e.g., volumetric data and polygon meshes, which is the data that is used to create posters with our framework, the illustrative visualization field has made steps to automate and replicate the work of scientific illustrators. An in depth discussion on illustrative visualization is presented in Section 2.2.

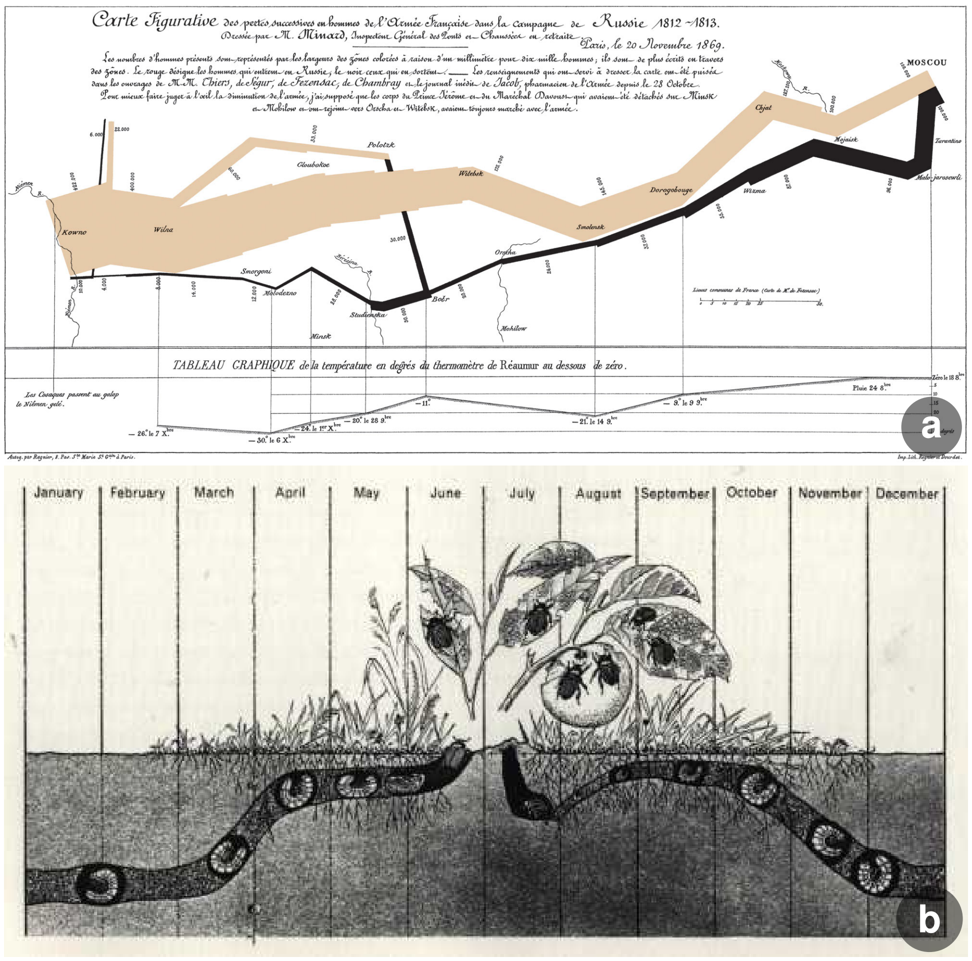

Carte Figurative des pertes successives en hommes de l'Armée Française dans la campagne de Russie 1812–1813.
Dressée par M. Minard, Inspecteur Général des Ponts et Chaussées en retraite.
Paris, le 20 Novembre 1869.
MOSCOU
TABLEAU GRAPHIQUE de la température en degrés du thermomètre de Réaumur au dessous de zéro.
a
January
February
March
April
May
June
July
August
September
October
November
December
b

Figure 2.2: Examples from the Graphical Excellence chapter in Tufte's book: *The Visual Display of Quantitative Information*. Minards figurative map (a) from *Tableaux Graphiques et Cartes Figuratives de M. Minard* showing the movements of Napoleon's troupes in Russia, and the life cycle of the Japanese beetle (b) from *Man and Insects* by L. Hugh Newman.

Engagement

The second factor mentioned by Kosara for creating successful communicative visualizations is *engagement*. There are different ways of creating engaging visualizations, e.g., according to Moere and Purchase, aesthetic representations can lead to better user engagement which in turn can lead to more effective communication [73]. Another way of enhancing engagement is to create a narrative around the data. ElShafie found that constructing a story to tell with the data can ease the comprehension of complex scientific topics for a general audience [29]. She also emphasizes the necessity of knowing one's audience. The goals of communication are generally different when addressing fellow domain experts versus the public, e.g., explaining detailed scientific results and methods versus providing an engaging overview of the topic of research to spark interest. Thus, for the general public, a story could be an appropriate means of communication, whereas it could be distracting when communicating with fellow domain experts.

Chu et al. affirm that presenting data as a narrative can contribute to efficient communication by providing new perspectives on the data which can lead to new insights [23]. When creating narrative visualizations, traditional rhetoric techniques of logos, ethos, and pathos can be used to invoke stronger engagement on the message of the visualization, or to imply certain interpretations of the data [43]. Communicating data through a narrative can, in addition to creating engagement, make the message more memorable and easier to comprehend for a general audience [55]. Our framework creates narrative visualizations that can tell a story through the sequence of scenes in the poster. The topic of narrative visualization is covered in Section 2.3.

2.2 Illustrative Visualization

Scientific illustrations have been used to communicate and educate for centuries. Leonardo da Vinci is possibly the most famous scientific illustrator and works such as the Vitruvian Man, shown in Figure 2.3 a, are iconic more than 500 years after his death. Scientific illustrations include directions such as medical illustration, Figure 2.3 b, and technical illustration, Figure 2.3 c. The illustrations are abstracted from the real-world objects they depict, yet are usually highly detailed and require a substantial amount of manual labour to create. Traditionally this was done by drawing on paper, but more recently digital tools such as Photoshop [4] and After Effects [3] have simplified the process. Illustrative visualization combines centuries of knowledge from the traditional illustration domain with modern algorithmic approaches and graphical hardware to more or less automatically generate illustrations from data.

A common technique in scientific illustration is abstraction from reality. The illustrations are not meant to be photorealistic, i.e., they do not try to depict the object or scene as it could be seen in reality, but are simplified or distorted so that the message presented by the visualization is in focus. Garrison et al. define a 2D abstraction space for biomedical communicative visualizations where the dimensions are model abstraction and visual abstraction [36]. The model abstraction relates to the level of detail in the communicated knowledge, whereas the visual abstraction relates to the visual complexity. Traditional illustrations can lie anywhere in this abstraction space, the model abstraction

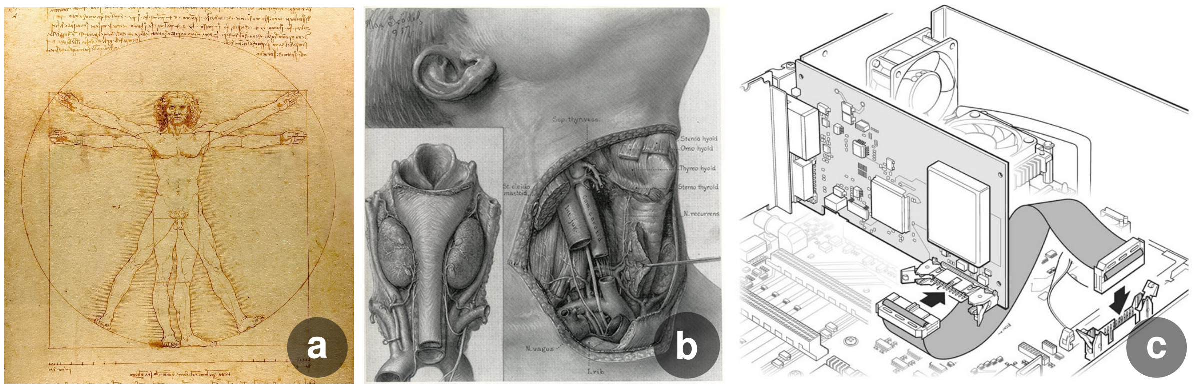


Figure 2.3: Examples of scientific illustrations. The Vitruvian Man by Leonardo da Vinci (a) and an illustration of the human throat by Max Brödel (b) are examples of traditional medical illustrations. The last image (c) depicts a PC interface card [38] and is an example of a technical illustration with a more modern style.

can be high, e.g., the beetle life cycle in Figure 2.2 b, or low, e.g., the PC interface card in Figure 2.3 c. Likewise, the visual abstraction can be high, e.g., the map of Napoleons troops in Figure 2.2 a, or low, e.g., Figure 2.3 b. In this abstraction space, posters created with our framework present a low level of detail, i.e., they are generally high in the model abstraction dimension, whereas they would generally be low in the visual abstraction dimension since the 3D models are rendered with a realistic shading model.

Another definition of abstractions in illustrative visualization was provided by Rautek et al. as a distinction between how to render and what to render [65]. Low-level abstraction, i.e., how to render, encompasses algorithmic techniques to render 3D structures in an artistic and non-photorealistic way. Examples of these techniques are described in Section 2.2.1. High-level abstraction, i.e., what to render, includes strategies to emphasize salient structures and remove or de-emphasize less important structures, is detailed in Section 2.2.2

2.2.1 Low-Level Visual Abstractions

On the lowest level of abstractions from illustrative visualization we can consider *how* different components are visualized. Illustrative visualization and non-photorealistic rendering techniques exist to emphasize important features of a model or a scene while reducing visual complexity and preserving the context around the features of interest. These techniques are often inspired by traditional illustration and seek to recreate the effects of this manual work in an algorithmic way. A great number of these techniques have been developed and covering them comprehensively is not feasible in the scope of

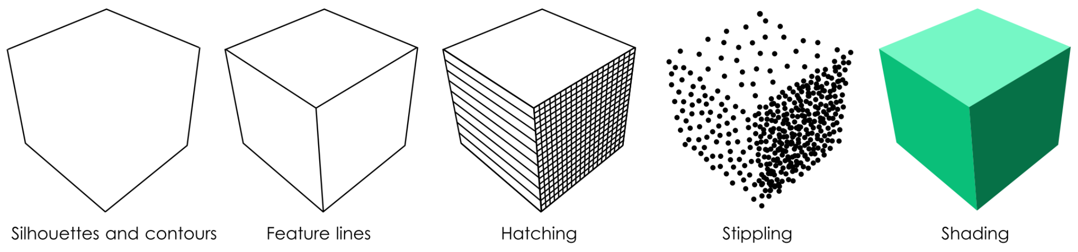


Figure 2.4: Categories for 3D surface-based illustrative rendering as presented by Lawonn et al. [52]

this thesis. We attempt to create an overview with the goal of providing context around our contribution.

In their survey, Lawonn et al. present an overview of 3D surface-based illustrative rendering techniques [52], all of which are part of the low-level visual abstractions. They split the techniques into five categories, silhouettes and contours, feature lines, hatching, stippling, and illustrative shading, as illustrated in Figure 2.4. Each of these categories contain multiple techniques and variations. Silhouettes and contours, in addition to feature lines, draw inspiration from traditional sparse line drawings which aim to give a simplified view of a geometry by drawing only a limited number of lines. Csebfalvi et al. present a technique for generating and rendering contours from volume data [26]. Thickness of contour lines from 3D surface data, i.e., volumes or polygon meshes, are optimized by Kindlmann et al. in their work on transfer functions based on second order derivatives [47]. Both of these works are in the category of silhouettes and contours, although the technique of Kindlmann et al. could be adapted to optimize feature lines instead of contours.

Hatching and stippling make use of surface-filling marks to create a shading effect which shows the 3D structure of a surface in an abstracted way. Zander et al. propose a method to create a hatching visualization based on polygon meshes [79], whereas Aidong et al. base their stippling visualizations on volume data [54].

Illustrative shading is the least abstracted category and includes inspiration from comics and cartoons, traditional illustrations, and more. Techniques that, for example, render 3D objects in a painted style, e.g., watercolor, would be placed in this category [10, 21]. Gooch et al. present a non-photorealistic lighting model inspired by technical illustration which is also placed in the illustrative shading category. The technique uses both hue and luminance, instead of solely luminance as in classic Phong shading, to convey spatial information [39]. The benefit of this technique is that extreme luminances, i.e., very light or very dark colors, are not used by this shading and can rather be used to enhance a visualization with classical illustration techniques such as outlines, with very dark colors, and highlights, with very light colors. In a related work, Wang et al. created a system to help users pick meaningful colors for their visualizations with an emphasis on contrast [75]. They consider two levels of contrast, hue and luminance, which are linked to inter- and intra-class contrast respectively. In addition, vividness of a color is mapped to importance.

Another important technique in traditional illustration is deformation of surfaces, e.g., by enlarging important structures and reducing context structures. This has, for example, been done by Reinert et al. inspired by the sensory and motor homunculi, where annotations of importance provided as input determines the size of the deformation [7]. An example of this technique is shown in Figure 2.5. This type of deformation based on importance can be found in traditional illustration all the way back to ancient Egypt where size of figures in an engraving denotes their significance. Another approach by Keahey and Robertson deals with deformation in image space by magnifying regions and creating smooth transitions between the magnified and non-magnified regions in the image [45]. For both of these approaches, the goal of the deformation is to make the most salient features of an object larger to emphasize them. Yu et al. create distortions in objects by combining multiple perspectives [78] and their approach can be used to, for example, create different postures from the same static model. A different usage for deformation of surfaces or structures is to avoid occlusion of an object or structure of interest. This could for example be done in a view dependent manner by distorting

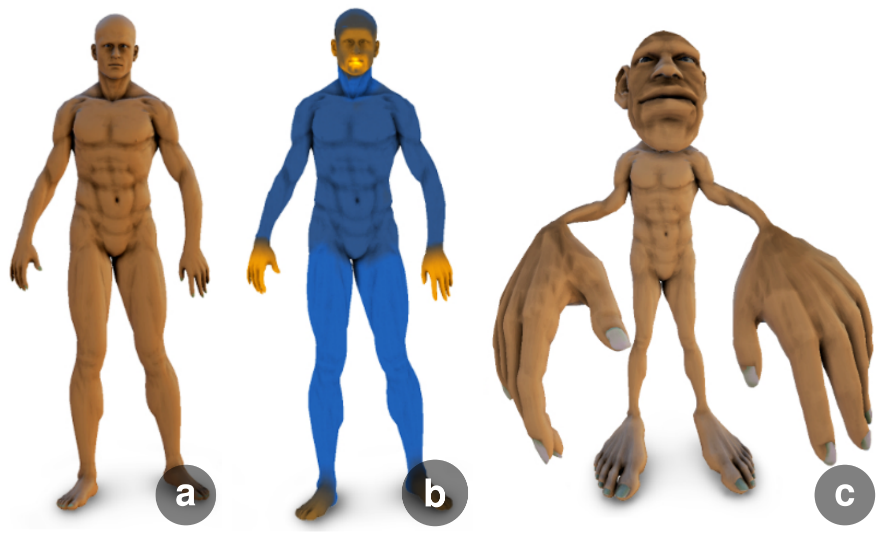


Figure 2.5: Example of the warping technique developed by Reinert et al. [7], showing the original model (a), the importance mask (b) where the lighter yellow is more important, and the resulting deformed model (c).

structures that intersect with a viewing ray from the camera to the object of interest [18]. In our framework, we use deformation to smoothly connect neighbouring scenes with the focus object, in addition to making important scenes appear larger in the final poster.

Low-level illustrative visualization abstractions can also be optimized in a domain specific way. For example, the techniques and ideas have been used in flow visualization [19, 11] to create simplified images. However, the abstracted representations of the flow, e.g., an iso-surface, can turn out to be very complex and the shape can be difficult to perceive, especially when using transparency. Carnecky et al. solve this by using cues from technical illustration to enhance perception of these complex transparent surfaces [17].

Frameworks to combine multiple illustration techniques to meet different tasks and objectives have been developed. Bruckner et al. present a framework that can composite multiple rendered layers of an object or scene with different blending operators and layer masks [15]. This allows users to create illustrative visualizations with traditional illustration effects such as selective transparency.

2.2.2 High-Level Visual Abstractions

In addition to the low-level abstractions of *how* to renders models or scenes, we must consider *what* to render through high-level abstractions. This is important to simplify visualizations and focus the viewer's attention on the most important aspects of the model or scene. Usually we want to keep the less important structures or features in the visualization, but de-emphasized, to provide context while focusing the viewer's attention on salient features. This type of visualization is called focus+context.

One approach to focus+context visualizations is using smart visibility, i.e., cut-aways, ghosting, or exploded views [74]. An example of how a cut-away technique is used in a traditional illustration can be seen in Figure 2.3 b. Cut-aways refer to views where parts of an, or multiple, occluding structures are removed to reveal salient structures behind that would otherwise not be visible, e.g., removing skin to view muscle tissue underneath. Different types of cut-away techniques exist. Correa et al. use metaphors from medical treatment, with tools such as peelers, retractors, pliers and dilators, to create specialized cut-away visualizations with volumetric data [24]. Users can choose to align the cuts along an axis, a surface, or a segment to produce the desired effect. Ghosting refers to a technique where occluding structures are drawn in an abstracted way, e.g., low opacity, line drawing, etc., while the otherwise hidden salient structure is drawn in full detail and opacity. Bruckner et al. present a ghosting-inspired technique for volumetric data [14]. An example of this is shown in Figure 2.6 a. Krüger et al. publish, in the same year, a very similar technique where transparency and shading are used to create visualizations that are close to the illustrations using ghosting [51]. This technique can, however, be applied to polygon meshes in addition to volumetric data. Finally, exploded views refer to visualizations where objects are segmented and these segments are displaced based on, for example, physics inspired forces [13] as shown in Figure 2.6 b. Efforts have been made to combine smart visibility techniques in an intuitive way. In VolumeShop, Bruckner and Gröller combine non-photorealistic rendering with cut-away and ghosting techniques in one unified system [12]. They can generate static images that bear resemblance to traditional illustrations with the benefits this entails, while still allowing for interactive manipulation of the images.

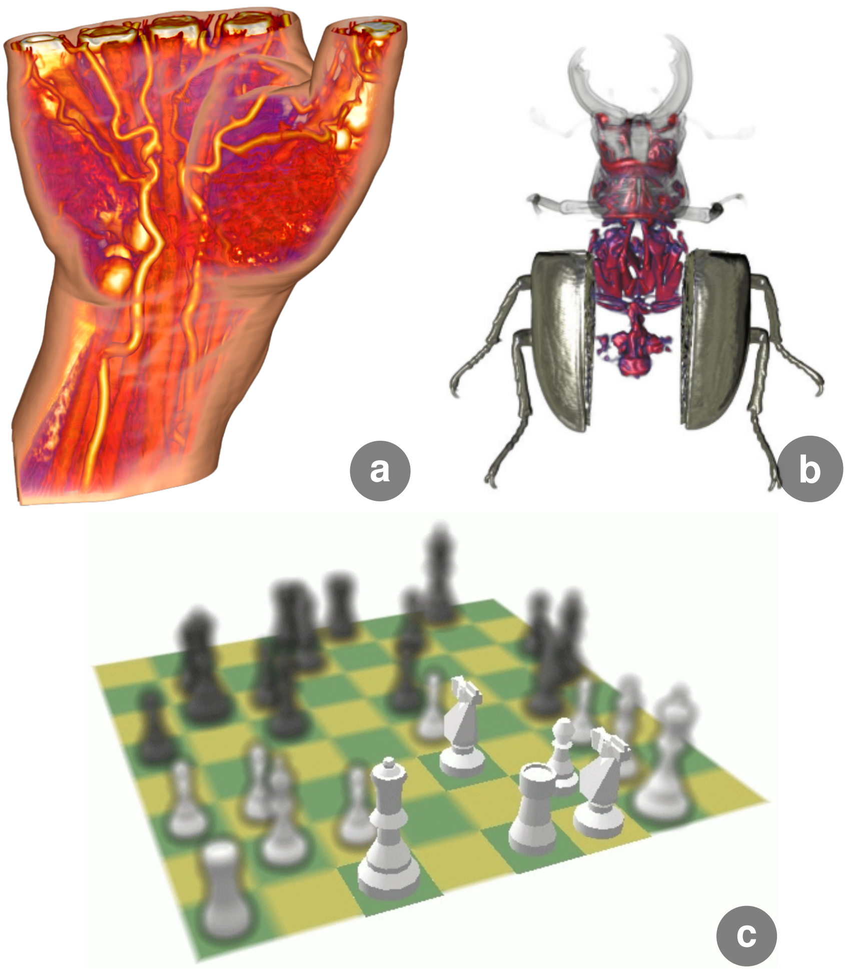


Figure 2.6: Examples of high-level visual abstraction techniques: ghosting (a) by Bruckner et al. [14], exploded views (b) by Bruckner and Gröller [13], and semantic depth of field (c) by Kosara et al. [48].

Kosara et al. present a novel focus+context visualization called Semantic Depth of Field [48]. In contrast to traditional depth of field that decides the level of blurring based on the distance to the camera, they use blurring to de-emphasize the context structures and render the focused objects or structures sharply, for example, as shown in Figure 2.6 c.

Another problem which is solved in scientific illustration, but which is non-trivial with common visualization techniques, is depicting data with scales on different orders of magnitude, often referred to as multiscale data, in the same visualization. This is useful in, for example, medical illustration where one might want to show whole organisms, e.g., humans, organs, cells, and molecules in the same image. An illustrator can easily change the scale of structures in the image artificially and blend between different scales, an award-winning example is shown in Figure 2.7 a. But when basing a visualization on real-world data, this is a challenge. Halladjian et al. solved this challenge by smooth transitions between zoom levels as a viewer zooms in or out on a multiscale structure, e.g., a DNA molecule [40]. In their proposed framework, an abstracted 2D representation of the higher scaled data is embedded semi-transparently in the 3D representation of the currently focused scale level. The higher scaled data is embedded to provide context. A downside to this approach is that all data can not be viewed simultaneously, although the benefit is that they achieve a more accurate depiction of how the different scales relate to each other, also spatially. A different approach to visualizing multiscale data, which is closer to traditional illustrative approaches, is presented by Hsu et al. [42]. They create visualizations that seamlessly combine multiple scale levels of a 3D model. The user sets up pinhole cameras at different scales to render desired parts of the model. The framework combines the rendering of each camera into one final continuous image by using non-linear viewing rays. Our framework also combines the rendering of multiple pinhole cameras, i.e., perspective cameras, but this is done with depth composition and opacity manipulation rather than non-linear viewing rays.

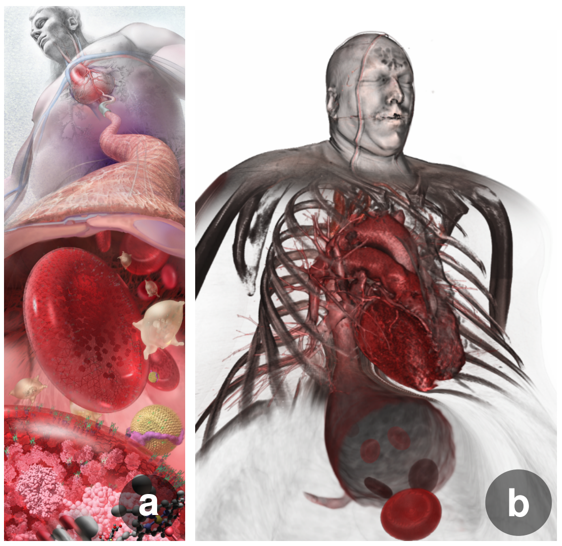
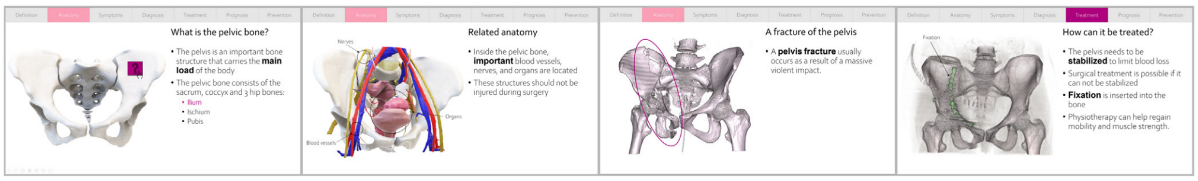


Figure 2.7: A multiscale illustration (a) *Zoom Into the Human Bloodstream* by Linda Nye and the Exploratorium Visualization Laboratory [60], and (b) the multiscale visualization by Hsu et al. inspired by the illustration [42].

2.3 Narrative Visualization

A narrative is defined as a sequence of chronological events that are linked in some way, whereas a story has a plot with a setting and a resolution, according to ElShafie [29]. With this definition, narrative is a broader term with stories as a subset, where the sequence of events form a plot. Others provide a circular definition of the terms, i.e., a story is a narrative and a narrative is a story. We will use the definition by ElShafie in this thesis because it is useful to distinguish between the two terms for our framework. The framework creates posters with a sequence of scenes, i.e., a narrative. Depending on what these scenes are depicting, the sequence can form a plot, and thus a story, or not. The responsibility of creating a story lies with the users of our framework, whereas we can only guarantee the creation of a narrative.

In addition to providing these definitions of narrative and story, ElShafie describes what a good science story is [29]. For example, it is important to find the story in the data, rather than forcefully fitting the data to a pre-determined story. Finding stories in data is not trivial, and it is not given that there are stories to find. This difficulty is one of the main challenges of visual storytelling, according to Ma et al., and solving it requires collaboration between domain and visualization experts [55]. Other considerations to make when creating science stories include the audience of the story. The level of knowledge on the subject of the story is a crucial factor, and should guide the creation of the visual science story [55]. Stories might not be an appropriate communication form for all audiences, e.g., fellow domain experts, but can be used to bridge the gap between general audiences and complex scientific topics [29]. An example of this is provided by Meuschke et al. in their work to communicate disease data through narrative visualization primarily to a non-expert audience with and interest in medicine [56]. Diseases, including symptoms, diagnosis, treatment, prognosis, and possible prevention, are complex scientific topics and are difficult to understand for the general public with little medical knowledge. They compared their interactive narrative visualization, an example is shown in Figure 2.8, to a more traditional blog design, and found that their approach increased both memorability and engagement on the topic. Although finding a story to tell and considering the intended audience are clearly important for a successful narrative visualization, they lie outside the scope of this project and are the responsibility of the author of a poster.



What is the pelvic bone?

- The pelvis is an important bone structure that carries the **main load** of the body
- The pelvic bone consists of the sacrum, coccyx and 3 hip bones:
  - Ilium
  - Ischium
  - Pubis

Related anatomy

- Inside the pelvic bone, **important** blood vessels, nerves, and organs are located
- These structures should not be injured during surgery

A fracture of the pelvis

- A **pelvis fracture** usually occurs as a result of a massive violent impact.

How can it be treated?

- The pelvis needs to be **stabilized** to limit blood loss
- Surgical treatment is possible if it can not be stabilized
- **Fixation** is inserted into the bone
- Physiotherapy can help regain mobility and muscle strength.

Figure 2.8: Example of the narrative visualizations created by Meuschke et al. to communicate disease data [56]. The figure shows only an excerpt of the visualizations.

Design Space of Narrative Visualizations

Efforts have been made to define the design space of narrative visualizations. Segel and Heer propose a design space with three dimensions: genre, visual narrative, and visual structure [68]. The design space was based on an analysis of 58 real-world narrative visualizations mainly from journalism. According to the authors, there are seven genres of narrative visualizations: magazine style, annotated chart, partitioned poster, flow chart, comic strip, slide show, film/video/animation, as shown in Figure 2.9. The authors acknowledge that for more complex visualizations, multiple genres must be combined. The map by Minard from Figure 2.2, for example, does not fit into any one of these genres although it is an established example of a narrative visualization. It could, however, fit into a combination of the *annotated chart*, *partitioned poster*, and *flow chart* genres. TimeBender posters could be defined as a combination of the *annotated chart*, *partitioned poster*, *flow chart*, and *comic strip* genres.

The second dimension, visual narrative, relates to the visual expression supporting the narrative. This dimension is divided into three categories: visual structuring, high-

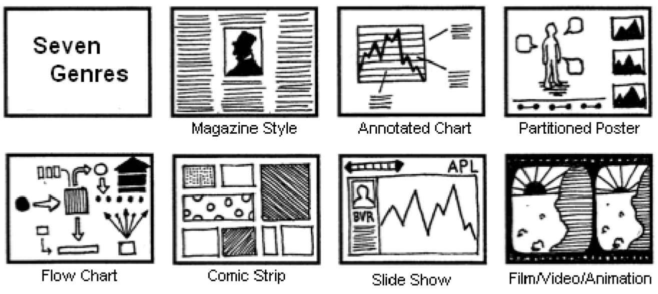


Figure 2.9: The genres of narrative visualizations as defined by Segel and Heer [68].

lighting, and transition guidance. Visual structuring means that visual elements are used to show an overview of the narrative and to place the user at their current location in the narrative. Examples of visual structuring include progress bars and check lists. Our framework does not include any visual structuring elements as the viewer of a poster created with our framework can always see an overview of the poster since the whole poster is visible at all times. Highlighting means that visual elements are used to help focus the audience's attention at important features of the narrative. Highlighting is similar to techniques used in focus+context visualizations, as discussed in Section 2.2.2. Examples include increasing size, using a contrasting color, and blurring objects that are not in focus. Highlighting is included in our framework by allowing the author to decide the size of each scene in the poster such that important scenes can be larger in size. The author could also, for example, use vibrant colors on the focus object to make it stand out. Transition guidance emphasizes the importance of ordering in a narrative. Techniques for transition guidance are designed for smoothly switching between subsequent scenes and include animated transitions and object continuity. In TimeBender, the transition between scenes is guided by object continuity, as the focus object continuously ties subsequent scenes together.

Narrative structure, the third dimension of the design space defined by Segel and Heer, incorporates all non-visual narrative aids. This dimension is also divided into three categories: ordering, interactivity, and messaging. Ordering indicates which structure the narrative has, e.g., author-driven or user-driven. Interactivity, as the name suggests, indicates how the user can interact with the narrative, e.g., moving between scenes, accessing details, filtering data, etc. Messaging describes textual commentary to the visualization. This can be a title, short textual annotations, or longer explanations in, for example, a news article. In our framework, we use a author driven, i.e., linear, ordering of the narrative. In addition messaging can be added in the form of titles and short annotations to the scenes. Audience interactivity was not a focus during this work, but it an avenue for future work, as will be explained in Section 7.

Narrative Visualization Approaches

Although narrative visualization is a relatively young research field, many techniques, frameworks, and tools have already been published. Wohlfart and Hauser proposed a framework that focuses of narrative visualization for volume data [77]. The narratives are created by setting up key frames which should show the main interesting features that the author wants to focus on. When viewing the visualization, the key frames are shown in sequence with smooth animated transitions. The goal of the smooth animation is to keep the user's mental map intact between each key frame. The framework also allows for limited interactivity during the viewing of the visualization. The audience can for example change the viewpoint in a key frame. Wohlfart and Hauser argue that allowing a viewer to interact with a narrative visualization will lead to the viewer trusting the results shown in the visualization more as they can verify the findings [77]. However, the interactions can also introduce additional complexity, for example, letting the audience change the viewpoint requires that they are able to navigate in a 3D environment, which is not trivial.

Visual storytelling has been proposed as a way to bridge the gap between visual analytics, which can involve mentally combining results from multiple complex visualization, and general audiences. According to Chen et al., creating a story around analysis results can help audiences without visual analytics experience or knowledge to understand the results and their implications [20].

Comics-based storytelling [23] is an approach that draws inspiration from comic books, where scenes are layouted on pages in a sequence that creates a narrative. The scenes can be annotated with speech or though bubbles, or short explanatory texts. Scene importance can be encoded in the space allocated to the scene within the comic-book page. The narrative structure is usually encoded implicitly with the page allocation and the layout within the page. TimeBender has similarities to comics-based techniques. Individual scenes are set up and tied together in a narrative. The importance of a scene in our tool can also be encoded by the size of the scene. However, the narrative structure in posters generated with our framework is explicitly shown by the focus object which means that the layout of the poster is not constrained by the need to create an implicitly understandable narrative flow.

Correa and Ma present a technique that, in the resulting visualizations, carries some similarities to comics-based storytelling, dynamic video narratives [25]. They create a summary of videos in one static image by combining scenes consisting of multiple frames on a continuous background. In Figure 2.10 a, an example created from a Superman cartoon is shown. By combining multiple frames a motion blur effect is created to show movement and speed. In the resulting visualizations, time is implicitly moving from left to right along the vertical axis and all narratives are linear in time. The continuous background creates a coherent summary image. We share a similar goal of creating a coherent narrative visualization. In contrast to Correa and Ma's solution, we solve the coherency by connecting all scenes with a common "red thread", the focus object. This allows our resulting images to be more flexible when it comes to arranging the scenes as the time, or order, of the narrative is encoded with the focus object, and not implicitly from left to right along the vertical axis. Scenes can therefore be placed however the user wishes while preserving the ordering of scenes. This can be beneficial, for example, if the author wants to facilitate the comparison of two scenes that are not neighbors in time.

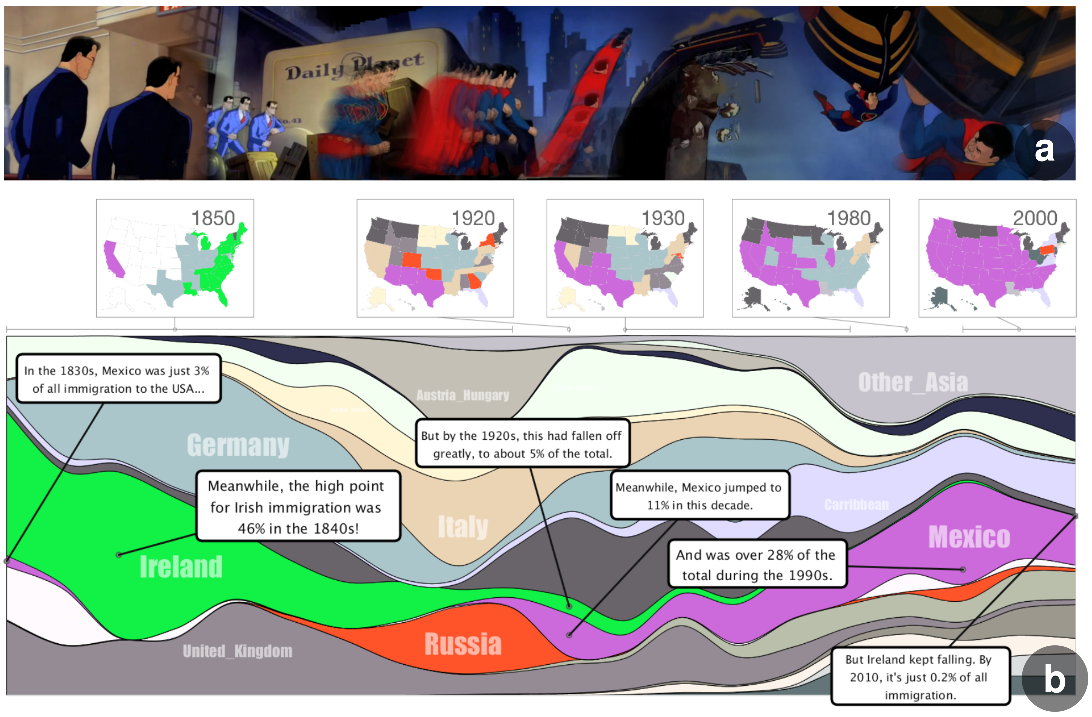


Figure 2.10: An example of a dynamic video narrative [25] depicting a scene from a Superman cartoon (a), and a Temporal Summary Image [16] showing the nationalities of immigrant to USA, as well as their geographical locations at selected years (b).

Another related approach was presented by Bryan et al. as a framework called Temporal Summary Images (TSI) [16]. The framework includes support for analysing data to find interesting story points, and creating a presentable narrative visualization based on these. The final image combines one continuous view of a selected data feature as a function of time, snapshots at interesting points in time showing different aspects of the data, and textual annotations. Figure 2.10 b shows an example of a TSI depicting the distribution of nationalities of immigrants to the United States of America with snap shots showing their geographical distribution in the country. Annotations guide the audience through the narrative, contrasting the immigration if Irish and Mexican citizens. In this approach the snapshots, which can be considered as scenes in a narrative, are tied together by the continuous temporal view of the data. The narrative in the TSI is implicit and must be outlined by textual annotations. The type of data is restricted to data with two main dimensions where one is temporal. In the example in Figure 2.10 b, the second main dimension is nationality of the immigrants.

Chapter 3

Methodology

The goal of TimeBender is to aid the authoring of posters in a 3D environment with a space-time dimension. The time dimension follows an elongated 3D model, called the *focus object*, that connects, as well as participates in, the scenes of the poster. The focus object is modeled as a spline to ensure smoothness and continuity throughout the poster.

Authoring of such posters is carried out through a five step pipeline. The author first devises a narrative, i.e., a sequence of scenes, that will showcase insights about the data that they want to communicate with the poster. This can be done by distilling a story from data as suggested by ElShafie [29], i.e., finding a story in the data rather than creating a story around the data. Second, each individual scene is set up in what we have called the local layout step. Third, in the global layout step, the scenes are positioned in the poster space, similar to layouting in a comics-based visualization [23]. Fourth, the detailed layout creates the final poster with textual elements. And finally, the author presents the poster to a target audience. As discussed in Section 2.1, visual communication, notably through visual storytelling, must be targeted towards the intended audience to be effective, e.g., by deciding the level of detail in the visualization based on the prior knowledge of the audience. The scope of our framework includes the three middle steps, i.e., the local, global and detailed layout, which correspond to the creation of a poster. The first and last steps remain the responsibility of the author and will not be discussed in this thesis. In this chapter we provide a description of the framework, including the three middle steps of the authoring pipeline, and functionalities that are not tied to a specific step of the pipeline, i.e., the modelling of the focus object, and soft fading, both in depth and edges of scenes.

3.1 Main Pipeline

We will now describe TimeBender's main pipeline which has three steps, the local layout, the global layout, and the detailed layout as can be seen in Figure 3.1. For each step we describe what the step does, why it is needed, and how it works on a concept level.

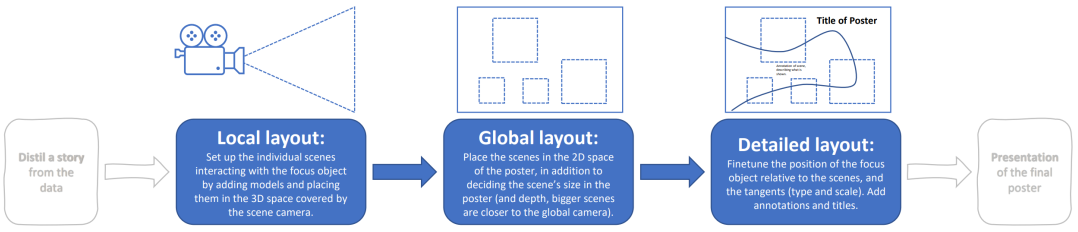

Title of Poster
Annotation of scene, describing what is shown.
Distil a story from the data
Local layout:
Set up the individual scenes interacting with the focus object by adding models and placing them in the 3D space covered by the scene camera.
Global layout:
Place the scenes in the 2D space of the poster, in addition to deciding the scene's size in the poster (and depth, bigger scenes are closer to the global camera).
Detailed layout:
Finetune the position of the focus object relative to the scenes, and the tangents (type and scale). Add annotations and titles.
Presentation of the final poster

Figure 3.1: An overview of the main pipeline of TimeBender. There are three main steps in the pipeline, local, global, and detailed layout. In addition to these we have two peripheral steps, not included in this framework but crucial to a successful poster, which are, first, to find a story to tell with your data, and finally to present the completed poster to an appropriate audience.

3.1.1 Local Layout

The first step of the pipeline is the local layout step. Here, each scene is set up individually by the author, i.e., one scene is viewed and handled at the time and no overview of the poster is given. Once each scene has been conceptualized and the author is satisfied with the appearances, they can move on to the next step. The reason for having the local layout as its own step, and not including this in the global layout step, is so that the author can focus on a scene with the whole screen allocated to the view of this scene. This lets the author see a larger and more detailed view of the current scene while preventing distractions from the other scenes. When allocating the whole screen to one scene, instead of sharing the screen with the rest of the poster, the author gets a clearer view with more details visible. Interactions with the models in the scene, e.g., moving, scaling, or rotating them, are easier since they appear larger on the screen. Viewing 3D scenes in this manner, i.e., one scene occupying the whole screen, is familiar to people, even without 3D modelling experience, as this is the common way to view 3D scenes in, for example, games and movies.

Each scene has a camera which captures what will be shown in the scene's section of the final poster. They also have a control point from the focus object spline with the spline segments on each side and the tangents determining the behaviour of the spline segments at the control point. In addition, the scene has an index which tells the framework the order of the scenes in the final poster, and a set of models that should be depicted in the scene. To create the local layout of a single scene, the author can add and remove models, and change their location, rotation, and scale. They can also change the 3D position of the control point of the focus object spline related to the viewed scene, as well as the tangents' orientation and scale. To ensure a smooth curve, the tangents entering and leaving the control point are coupled, i.e., they are enforced to be collinear by the framework, although the scale of the tangents may vary. The depth relationships between the models in the scene, including the focus object, are preserved in the final poster, which ensures that the intra-scene behaviour of the focus object spline remains close to the local layout.

To ease the authoring process during this step of the pipeline, the author can move the view, i.e., what is shown on the screen, around. This interaction, although introducing complexity by making the user navigate in a 3D environment, also has benefits such as making it easier to understand the depth relationships between models in the scene, or being able to get a more detailed view of certain aspects of the scene. To reduce the complexity slightly, we add the possibility to snap back to the view of the scene that will be portrayed in the poster, i.e., how the scene is viewed by the audience of the poster.

3.1.2 Global Layout

After finishing the local layout, the author enters the global layout step. Here, the author defines the layout of the scenes in the poster, which includes the 2D image position, i.e., at which width and height the scene should be drawn within the poster, and the size occupied by the scene, as shown in Figure 3.2. This step has similar considerations as comics-based visualizations, for example, as described by Chu et. al. [23], where the layout of scenes on a page can influence the flow of the narrative and the size of the scenes can indicate their importance to the narrative.

The global layout step is important because it defines the location and thus spacial proximity of scenes and the impression of importance, encoded by size, that a viewer gets when looking at the poster. Commonly in space-time narrative visualizations, time is encoded along the horizontal axis from left to right. Examples can be seen in Figure 2.10. This is also the case in comics-based visualizations where time is passing from left to right and top to bottom, assuming it is written in a language that is read from left to right.

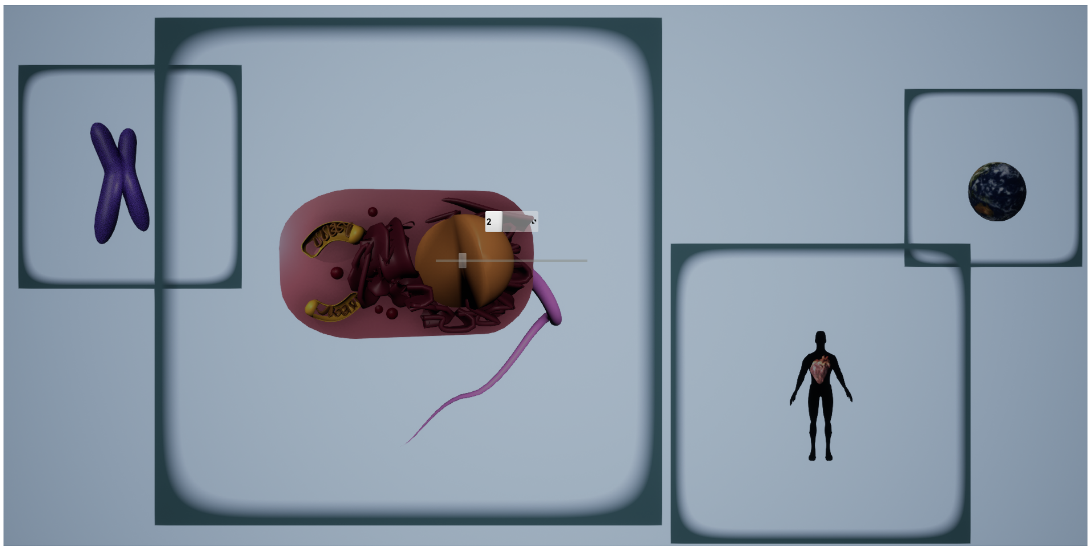


Figure 3.2: Example of the global layout step from the prototype implementation of TimeBender. The darker blue squares indicate the borders of the scenes, including a region of fading, as explained in Section 3.3. In the center, there is a slider which determines the size of a scene, the box above the slider indicates which scene's size is being altered.

Mangas, traditionally written in Japanese, are a well known counter example of this assumption. In our case, however, the temporal flow of the narrative is encoded with the focus object, i.e., the audience of the poster can follow the focus object from one scene to the next. This allows the author more freedom in the placement of scenes in the poster, and this placement can thus be used, for example, to simplify the comparison of scenes by placing them close together or indicating narrative closeness or distance with physical closeness or distance in the poster.

When entering this step of the pipeline, the author will initially be met with the scenes placed in an default layout. We chose a horizontal line where the size of the scenes is equal and determined by the number of scenes such that they all fit in one horizontal line. Since the author can decide how many scenes the poster should contain, we need an approach that is dependent on the number of scenes. We want to make sure that an arbitrary number of scenes fit into the view in a horizontal line. This is done by adapting the depth of the scenes: the more scenes, the less screen space each scene can take, and the deeper in camera space the scenes will be placed. Assuming that we know the width in world coordinates of a scene, we can calculate the needed width of the camera frustum, i.e., the volume of 3D space captured by the camera, by multiplying the width of one scene by the number of scenes. This width will be called w_f. The following equations are known relations between camera parameters based on the definitions of these parameters:

$$w_f = h_f \cdot a \tag{3.1}$$

$$h_f = d \cdot \tan \frac{fov}{2} \tag{3.2}$$

wherein h_f is the frustum height, d is the depth in camera space, fov is the field of view angle of the camera, and a is the aspect ratio of the camera. Since the camera parameters, a and fov, are known and we can calculate the desired w_f by multiplying the width of one scene by the number of scenes, we can combine and rewrite the equations to calculate the unknown, d.

$$d = \frac{w_f}{a \cdot \tan \frac{fov}{2}} \tag{3.3}$$

Solving for d gives us the depth from the camera that scenes must be placed at to fit in the view in a horizontal line. Because of the chosen horizontal alignment, the y-coordinate, i.e., vertical coordinate, in camera space of the scenes are the same, e.g., 0 for a horizontal line across the center of the view). The camera space x-coordinate, i.e., the horizontal coordinate, depends on the scene index so that the scenes are evenly spaced. The x-coordinate can be calculated as follows:

$$x_{sc} = \frac{1}{2} \cdot (w_{sc} - w_f) + i_{sc} \cdot w_{sc} \tag{3.4}$$

wherein x_{sc} is the camera space x-coordinate of the scene, i_{sc} is the index of the scene, and w_{sc} is the width of the scene. The term $\frac{1}{2}w_{sc}$ is added assuming the position of the scene corresponds to the center point of the scene. To add spacing between the scenes in the horizontal line, a larger value for w_{sc} can be used to calculate both the depth, d, and the x-coordinates, x_{sc}, of the scenes.

In the global layout, each scene shows the models as the author has set them up during the local layout step, but not the focus object which is added in the subsequent step. The scenes are shown with a border to easier see the size of the scenes relative to each other as shown in Figure 3.2. Without this border, the author would have only the size of the models to go by which is harder to compare, and each scene can have models of vastly different sizes. The border also helps determining the actual location of a scene in the poster as the models may not be populating the whole scene. To change the image position of a scene, the author can click and drag it to a chosen position in the poster. Clicking and dragging was selected as the interaction type as it is a common interaction for moving objects around a 2D screen and will thus be familiar to most users. Interacting in 3D is more difficult and is why we chose to abstract the third dimension, i.e., the depth, or distance, of a scene from the main camera, to simply the size of a scene. When moving a scene further away from the camera, it appears smaller, and vice versa. But the author does not need to be concerned about the depth, they are directly changing the size of a selected scene.

A consideration we must make when doing this abstraction is that if we only change the depth of the scene in camera space, and nothing else, the scene will move in image space since our camera uses a perspective projection. A perspective projection is similar to how we as humans view the world, and will create images where objects that are further away from the camera appear smaller, which is the desired outcome for our application. We must, therefore, adjust the horizontal and vertical position of the scene in camera space to compensate so that it seems to the author that the scene is stationary on the screen and simply increasing in size. To achieve this, we use the geometric concept of similar triangles. Similar triangles are triangles that have the same shape, i.e., the same angles, but not necessarily the same size. With two similar triangles, the ratio between corresponding sides are the same. We can use this knowledge to calculate the horizontal and vertical camera space position of a point in the scene that will remain fixed in image space, called a fixed point. This point can be chosen arbitrarily within the scene, but natural choices include the corners or the center of the scene, the latter was chosen for this framework.

We know the vertical position and depth of this fixed point before starting to change the depth of the scene. After changing the depth, we know the new depth but not the new vertical position of the fixed point in camera space. As shown in Figure 3.3, the camera and the two positions of the object form two similar right triangles. We know that they must be similar since they have the same angles: the angles at the camera are the same since this corner is shared by the two triangles and the angles where the depth and vertical axes meet are right, i.e., 90°. Since two of the angles are the same, the third must also be the same so that the angles add up to 180° as they must in a triangle. With two similar triangles, we know that the ratios between corresponding sides will be the same. Since we know the depth of each point, i.e., the length of the side of the triangles aligned with the depth axis, we can calculate this ratio, r, as follows:

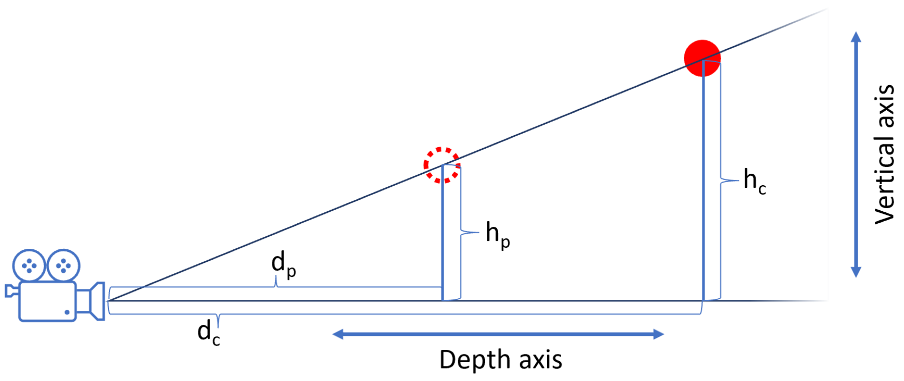


Figure 3.3: 2D illustration of the perspective projection showing only the depth and vertical axes. A similar figure could be made for the depth and horizontal axes. The stippled red circle is indicating the previous position of the chosen fixed point of the scene, and the red circle is the current position of this point. h_p, h_c, and d_p, d_c, are the previous and current height and depth positions of the point in camera space, respectively.

$$\frac{d_p}{d_c} = r = \frac{h_p}{h_c} \tag{3.5}$$

wherein the variables are defined in Figure 3.3. The unknown value in this equation is h_c, the vertical position of the fixed point in camera space after moving the scene in depth. Since the rest of the values are known, we can calculate the unknown, h_c, by rearranging the equation as follows:

$$h_c = \frac{h_p \cdot d_c}{d_p} \tag{3.6}$$

With this we have calculated the new vertical position of the scene, and the horizontal position can be calculated in the same way. This ensures that the chosen fixed point will remain in the same location in image space, and the scene will appear to remain stationary while growing in size.

3.1.3 Detailed Layout

The final step of the pipeline is the detailed layout. This step will produce the "camera ready" version of the poster, where all components: the scenes, the focus object, and the textual elements, are present. We add details in a final step because these are not important for the layout of the scenes in the poster, which is done in the global layout step, and can rather add distractions and reduce the efficiency of the author in this step. However, it is crucial that the author can add these details to the final poster before presenting it, to add explanation with annotations, to create interest and context with a title, and to define the compromise between intra- and inter-scene behaviour of the focus object, as well as the amount of distortions in the spline mesh. The compromise between intra- and inter-scene behaviour is necessary since the focus object is modelled as a spline. Therefore, the intra-scene behaviour of the focus object in one scene will be influenced by the positions of the neighbouring control points, and the direction and scale of their tangents. Since the position of a control point is determined by the position of the related scene in poster space, this also influences the intra-scene behaviour and can make it differ from the local layout. This can be counteracted in the scene by increasing the scale of its tangents, but doing this will influence the inter-scene behaviour creating a less smooth curve. The focus object can thus interact differently with the scenes than how it was set up during the local layout, but the author can get as close as they wish to the original setup while sacrificing the smoothness of the inter-scene sections of the focus object. Although the focus object can have a different intra-scene layout than in the local setup, the depth information from the local setup is preserved as shown in Figure 3.4.

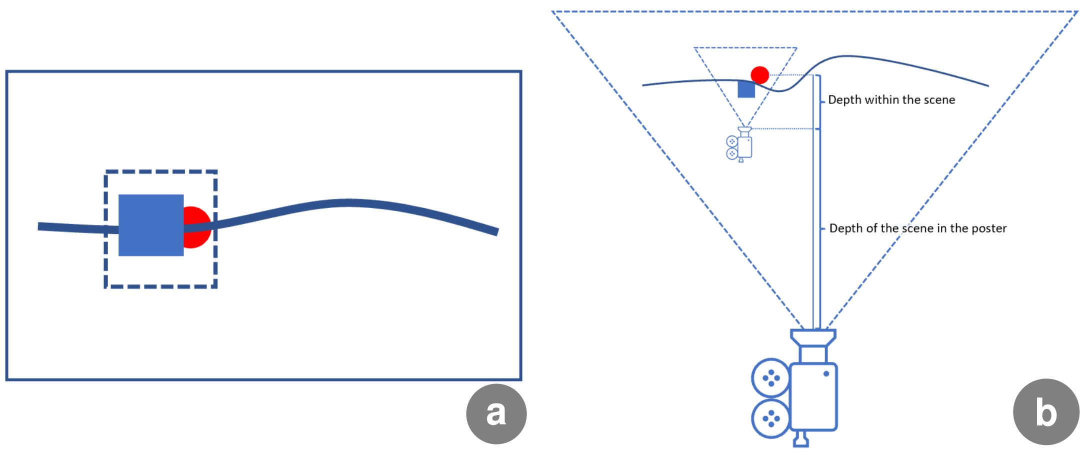


Figure 3.4: A simplified example of a poster with only one scene (a). The poster is a stitching together of views from individual scene cameras (only one in this case), showing models from the different scenes, and one main poster camera that is rendering the focus object (b). To stitch these together while conserving the depth relationships between the different components, we must combine the depth buffers of scene cameras with the main poster camera. The final depth of a pixel rendered by a scene camera in the main camera space is equal to the sum of the depth of the pixel in the scene camera space, and the depth of the scene in the main camera space.

The interactions in this step are similar to the interactions in the global layout step. The author can click and drag the control points to change their position in image space, i.e., in the poster. To change the scale of the tangents at a given control point, the author uses a slider (Figure 3.5 c). Additional interactions in this step include the author being able to choose between the two types of tangents: the self-defined ones from the local layout step, or tangents estimated using central differences based on the position in 3D camera space of the two neighbouring control points (Figure 3.5 a-b). A central differences approximation of a tangent t is calculated as follows:

$$t = \frac{p_{n+1} - p_{n-1}}{2\delta} \tag{3.7}$$

wherein p_n is the position of the n'th control point of the spline, and δ is the step size which we have chosen to be 1 by default. Each control point in the poster always has two neighbouring control points because additional control points are added to the start and end of the spline, as will be explained in Section 3.2.2 and shown in Figure 3.9. Because of this we do not need to make special considerations for the first and last scene of the poster.

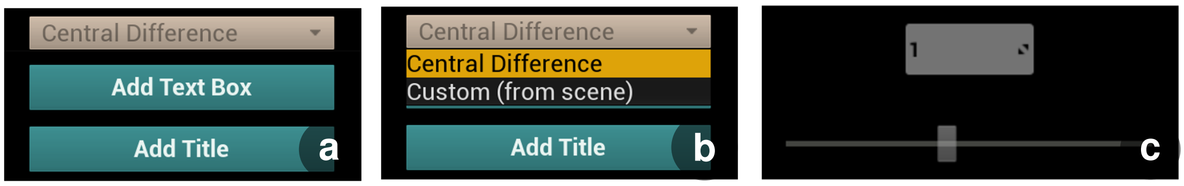
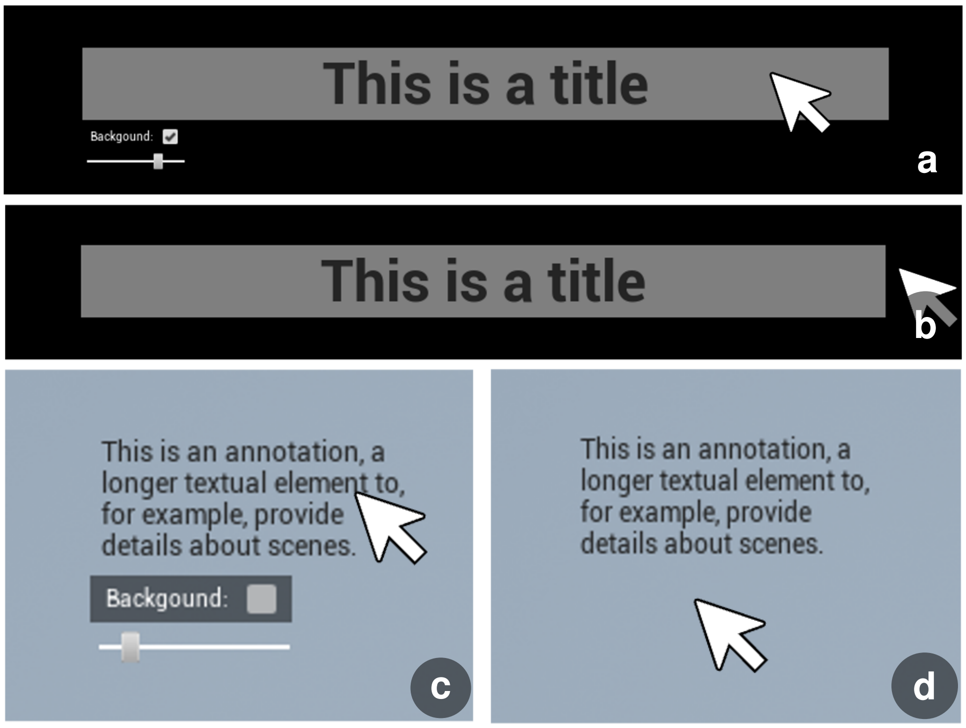


Figure 3.5: Graphical user interface from the detailed layout step in the prototype implementation of TimeBender. It shows the menu for selecting tangent type and buttons to add textual elements (a-b), as well as the slider to decide tangent scale (c). The box above the slider (c) determines which scene's tangents the scale is applied to, here it is scene 1.

The textual elements, annotations and titles, can enhance and guide the visualization by, for example, giving more information about what can be seen in the scene, or telling the viewer what to focus on within the scene. Titles can be used to create initial interest in the poster and to provide context for the visualization by telling the viewer the topic of the poster. Annotations can be used to provide additional details about a scene, or tell the viewers what to focus on in the scene. Textual elements are not necessary for all posters, e.g., if the poster is used only as part of a presentation and context, details, and guidance could be primarily provided verbally. By default, the annotations are textual elements with a small font without emphasis, a black color, and left justified text-box, see Figure 3.6 c-d, and the titles are in a large font with bold emphasis, a black color, and central justified text box, see Figure 3.6 a-b. This separation makes the information hierarchy clear for the viewer of the poster. The author can change the width of the text box which in turn will determine the height of the text box as the text is wrapped if it does not fit on a line. A background can be added to the text box, which ensures that the contrast to the text is high enough in any environment, e.g., as in Figure 3.6 a-b. And finally, the author can decide the location in image space of the textual element. This can be changed by clicking and dragging the text box to the desired location.



This is a title
Backgound:
a
This is a title
b
This is an annotation, a longer textual element to, for example, provide details about scenes.
Backgound:
c
This is an annotation, a longer textual element to, for example, provide details about scenes.
d

Figure 3.6: Example of textual elements from the prototype implementation of TimeBender. It shows a title (a-b) in a black environment with the contrasting background turned on, and an annotation (c-d) in a light blue environment. The interactable elements appear when the author hovers the mouse over the textual element (a, c), and disappears otherwise (b, d).

After tweaking the control points and tangents of the spline, and adding textual elements, the author has created the final poster. This can now be saved as a high resolution screen shot, i.e., the image is super sampled such that the resolution of the saved image can be larger than the resolution of the author's screen. In the case that the poster contains animated models, the poster can also be saved as a screen recording or presented through the framework.

3.2 Focus Object Approaches

As explained in Section 1, the poster depicts an elongated focus object with different scenes explaining aspects that form a narrative around it, e.g., a DNA molecule, a protein chain, a blood vessel, etc. Alternatively, the focus object can be a visual connection metaphor, e.g., a road, a river, a rope, etc., that supports the narrative flow of the poster. The author should be able to position their scenes arbitrarily within the poster to create the preferred global layout. This layouting flexibility is important for different reasons, examples include simplifying comparisons between scenes, indicating narrative closeness by physical closeness, highlighting narrative importance by physical size, or merely for aesthetic purposes. The challenge is to assure that the focus object acts as expected intra-scene, i.e., as the author has set it up within the individual scenes, as well as inter-scene, i.e., continuously and smoothly connecting the scenes in the correct order. We propose two approaches to achieve this, first by changing the viewpoint, i.e., the camera position and rotation, and second by deforming the mesh of the focus object. We first attempted the viewpoint-based approach with techniques from the computer vision field as we anticipated that it would lead to fewer distortions in the mesh. Unfortunately, this approach turned out to be sensitive to noise and not well suited to our specific usage. We, therefore, changed to a spline-based deformation approach which is what the final framework is built upon. Splines were used for deformation since they can produce smooth and continuous inter-scene behaviour, while allowing for detailed and customizable intra-scene behaviour with positioning of the control points and changing tangent directions and scales. In this section, we detail the two approaches and explain the encountered problems that led to the decision of using a spline-based deformation approach for handling the focus object.

3.2.1 Viewpoint-Based Approach

Initially, we drew inspiration from the computer vision field where there is a concept called relative pose recovery. The idea is to, based on two images of the same object or scene and matched points between the two images, recover the relative translation and rotation between the viewpoints of the two images. Figure 3.7 shows an illustratory example of how a pair of matched points is defined. The goal of pose recovery approaches is to find the rotation and translation required to move from the position of the camera viewing the first image to the position of the camera viewing the second image. We wanted to use relative pose recovery to determine the position of a camera viewing the focus object such that the image space locations of scenes in the poster would be as determined by the author.

An algorithm exists to perform this pose recovery with as little as five matched points, the appropriately named five-point algorithm by Nister [59]. This algorithm is based on

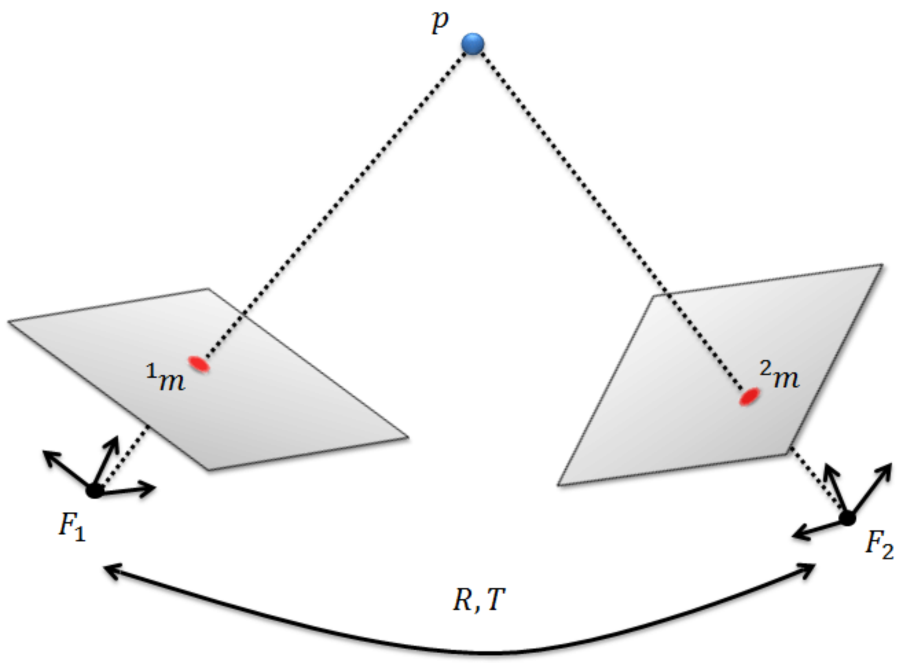


Figure 3.7: Two views, F_1 and F_2, of the same point, p, in a 3D scene [31]. Each view has a different position and rotation which changes the position of the point in each images, 1_m and 2_m. These image points are a pair of matched points, and the goal of relative pose recovery is to use matched points from the two images to derive the relative transformation from one view to the other. This includes rotation, R, and translation,T, of one view, e.g., F_2, relative to the other view, e.g., F_1.

linear algebra relations between the matched points and the camera parameters, such as their location and rotation. The five-point algorithm is non-trivial to implement and this is the reason why we instead used the related eight-point algorithm [41]. The eight-point algorithm is based on the same mathematical relations and produces similar results to the five-point algorithm with a less complicated implementation, although it is less computationally efficient. Since we were developing a prototype of the framework, the trade off between implementation complexity and computational efficiency was acceptable. In the following, we describe how we used the eight-point algorithm for our viewpoint-based approach, but the same concept, including the input design and the expected output, could be applied to the five-point algorithm.

The input to the eight-point algorithm is eight pairs of matched points given in normalized image coordinates, i.e., shifted from the range $[0, numberofpixels]$ to $[0, 1]$. To gather the input needed for the algorithm, the author of a poster would first set up scenes at interesting locations along the focus object (Figure 3.8 a), corresponding to the local layout step. They would then choose the 2D location of each scene on the poster (Figure 3.8 b), corresponding to the global layout step. Each scene corresponds to one matched point. These two steps would generate the two images with matched points needed for the eight-point algorithm. First, the locations of the scenes along the focus object in 3D space, as seen by a camera positioned at a default location, e.g., the origin. Second, an image of the selected locations in the 2D poster. Knowing the image coordinates of each scene in these two images, and assuming that the user places at least eight scenes, we have the needed input for the algorithm (Figure 3.8 c).

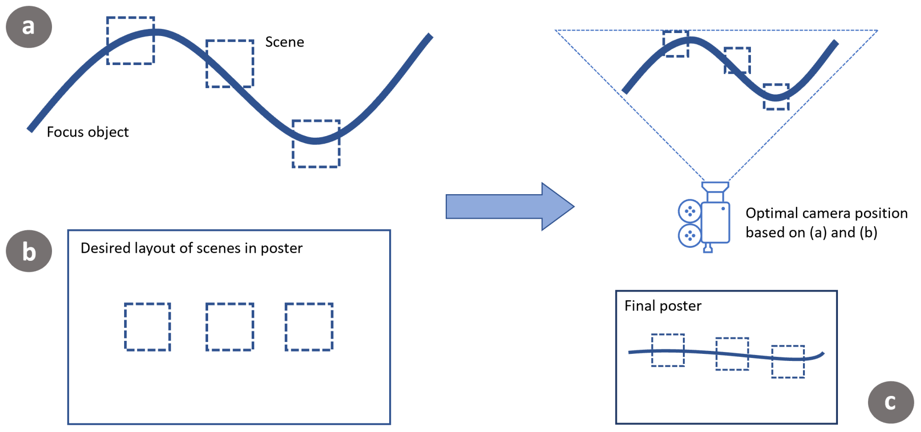


Figure 3.8: Illustration of the process of using pose recovery algorithms to find an optimal camera placement. First, (a) the author sets up scenes along the focus object, then (b) they define a desired layout of the scenes in the poster, in this example it is a simple horizontal line. Note that for the interest of simplicity and space, we have only included three scenes in this example, but at least eight would be needed. With the pose recovery algorithm, we find that the optimal position for the camera is to rotate it 90° around the horizontal axis to view the focus object from an angle where there is little vertical variation (c).

Ideally, the algorithm would then provide the best approximation of where the camera rendering the final poster should be positioned to view the scenes at the chosen locations in the poster. The benefits of a viewpoint-based approach is that it would introduce less distortions in the mesh of the focus object, as finding an optimal camera placement would limit the necessary amount of mesh distortions to achieve the intra- and inter-scene requirements. Unfortunately, the eight-point algorithm does not handle noise very well. Normally, one would have hundreds or thousands of matched points, and run this algorithm as the model in a RANSAC procedure [33]. This would ensure robustness against noise. We observed the poor handling of noise by testing different positions of the scenes in image space. When the scenes were positioned in a well-defined way, e.g., all points equally shifted or rotated, compared to their positions in the reference image the algorithm performed well, i.e., provided the expected rotation and translation. However, when arbitrarily positioning the scenes in the image space, or trying to move a scene from the well-defined position, the results from the algorithm became unstable. This is because there is no guarantee that a solution to the pose recovery problem exists when arbitrarily positioning the matched points in one of the images. The algorithm is designed for use on real-world images where there is always a well defined movement of the camera between the two images. In our scenario, the author must be able to place each scene where they wish in the poster which means that the resulting matched points do not necessarily correspond to a well defined camera transformation. In the case that they do not correspond to a well defined camera transformation, we can consider the distance between the author-defined matched points, and the closest set of matched points such that there exists a well defined camera transformation, as noise. Although there is noise in real-world images pairs also, this is tackled through the large number of matched points and the RANSAC algorithm. The eight-point algorithm cannot itself handle noise, which is why it turned out to be unstable in our scenario. We did not wish to compromise on the author's flexibility when positioning scenes, e.g., by enforcing that all scenes are shifted and rotated equally, which is why this viewpoint-based approach was discontinued.

3.2.2 Spline-Based Approach

When it became clear that the viewpoint-based approach would not yield the desired results because of the low number of sample points and high amount of noise due to the arbitrary placement of matched points, we decided to change to a spline-based approach. In this approach, the focus object is modelled as a spline, e.g., a Hermite spline or a piecewise Bézier spline. We decided to model the spline to be G_1 continuous, i.e., the arriving and leaving tangents at a control point have the same direction but not necessarily the same scale, with control points and tangents, although depending on the preferred type of spline, intermediate control point can be derived based on the tangents to yield an identical curve. In a poster with n scenes, the spline will have $n + 2$ control points for $n + 1$ segments as shown in Figure 3.9.

The number of control points and segments is larger than the number of scenes because there is one additional control point on each end of the spline. This allows for flexibility in the intra-scene behaviour of the focus object as each scene is related to one control point, connected to two spline segments, and this can be positioned by the author within the scene. In addition, the direction and scale of the arriving and leaving tangents of the control point can be specified. To ensure a smooth curve, the tangents are enforced to be collinear by the framework. The author can thus control the behaviour of the focus object within the scene, and the inter-scene behaviour is well defined, smooth and

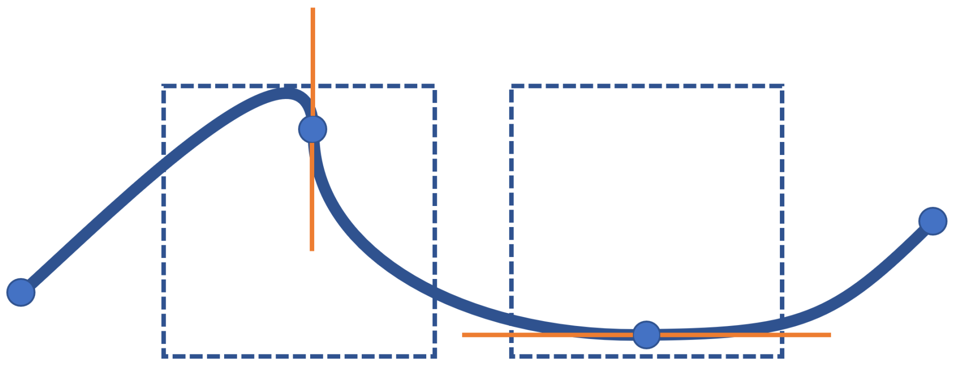


Figure 3.9: Schematic illustration of the spline-based approach for the focus object. The blue stippled squares indicate scenes ($n = 2$), the blue circles indicate spline control points ($n + 2 = 4$), and the curves connecting the control points are the spline segments ($n + 1 = 3$). As shown, the author can change the look of the focus object within a scene by the position of the control point, as well as the direction and scale of the tangents (shown as orange lines).

continuous, as a deformation of the focus object along the spline segment between the control points related to two neighbouring scenes. The inter-scene behaviour of the focus object can also be influenced by changing the tangents at the control points. A notable difference from the viewpoint-based approach, where camera placement and perspective were used to minimize the amount of deformation that the mesh of the focus object must undergo, is that this approach relies on mesh deformation along a spline that connects the scenes of the poster. This deformation can naturally cause stretching artifacts in the mesh. Using a mesh with a high number of polygons will yield better results.

3.3 Soft Fading Effects

To avoid harsh cuts at scene and poster borders, our framework includes fading at the edges of scenes as well as at the near and far clipping plane of the cameras. The near and far clipping planes are two imaginary planes that define the top and bottom of the view frustum of a perspective camera, i.e., at which interval of distances from the camera are objects captured. The view frustum is the volume that is captured by a perspective camera and is determined by different camera parameters, such as the field of view angle and the clipping planes, as shown in Figure 3.10.

To create a fading effect on the borders of a scene, a naive approach is to use a linear interpolation between 1 and 0 opacity starting at a desired distance from the scene

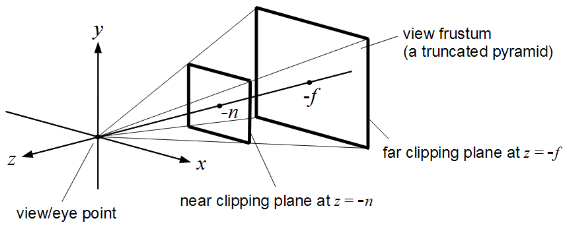


Figure 3.10: The volume called the view frustum is shown as the truncated pyramid indicated in the figure. As can be seen, the near and far clipping planes limit the view frustum in the top and bottom of the truncated pyramid shape [50].

border. The problem with this approach is the corners of the rectangular scene. In these corner regions, the interpolation creates a line artifact along the diagonal between the two edges which is not desirable. To counteract this, we employ superellipse geometry, also known as Lamé curves, as shown in Figure 3.2. A superellipse is defined as the set of points (x, y) that satisfy the following equation:

$$\left|\frac{x}{a}\right|^n + \left|\frac{y}{b}\right|^n = 1 \tag{3.8}$$

wherein a and b are floating point values having the property that for all points (x, y) that satisfy the equation, all x are within the interval $[-a, a]$ and all y are within the interval $[-b, b]$, i.e., the 2D points $(-a, -b)$ and (a, b) define an axis aligned bounding box of the curve. The parameter n in the equation determines the shape of the curve. If a point (x, y) satisfies the similar equation:

$$\left|\frac{x}{a}\right|^n + \left|\frac{y}{b}\right|^n = s \tag{3.9}$$

with $s < 1$ we know that this point is strictly inside the curve. Likewise, if a point (x, y) satisfies this equation with $s > 1$ we know that it is strictly outside the curve with higher values of s indicating greater distances from the curve. This information can be used to fade the edges of a rectangular scene. We perform the fading in image space with normalized image coordinates, i.e., the image coordinates are within the interval $[-1, 1]$. Firstly, we must determine the three parameters, a, b, and n. Setting $a = b = 1 - \delta$, with δ being a small positive number, will work if the scenes are quadratic, i.e., the width and height are equal. If the width and height are different, using these values for a and b will cause the fading region to be longer on the right and left, if the width is larger, or the top and bottom, if the height is larger, since we are using normalized image coordinates.

If $n = 2$ then the shape of the curve will be an ellipse, or a circle in the case that the width and height of the scene are the same (Figure 3.11 a-b). The larger n is, the more "squared" the shape will become, e.g., as seen in Figure 3.11. This could be a parameter that the user of the framework could set as it relies on preferences, but we did not include this possibility to reduce complexity. We recommend an n around 5 as a general setup which was selected empirically.

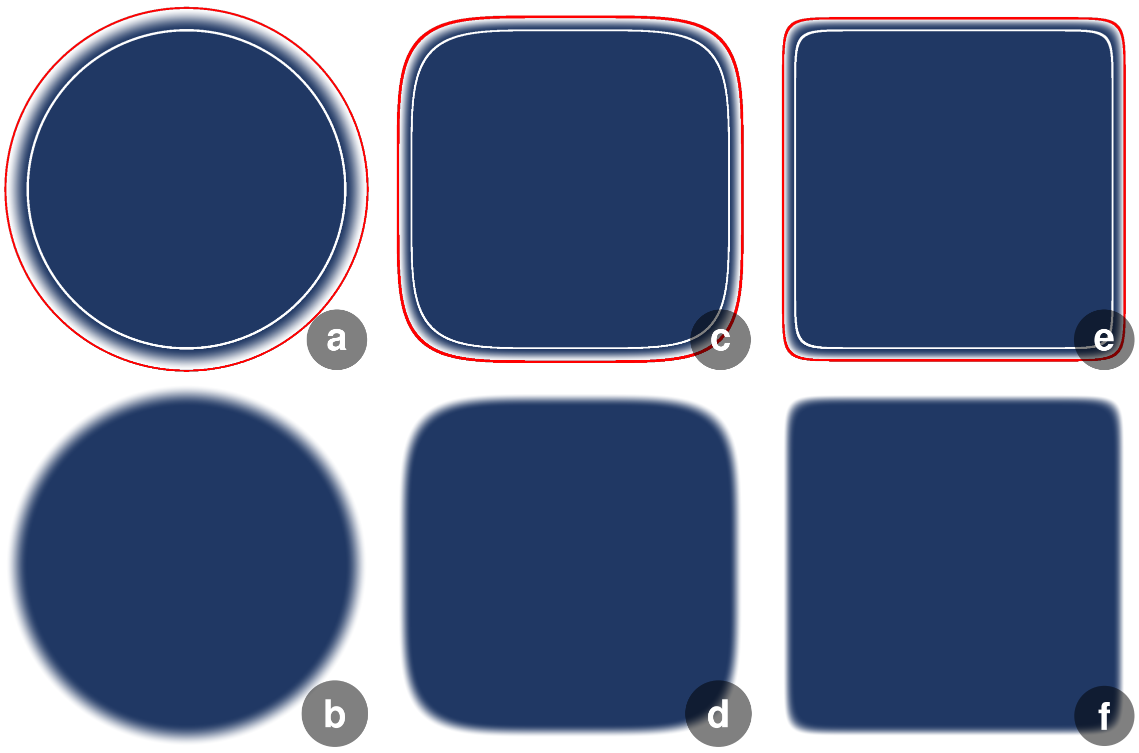


Figure 3.11: Superellipse fading with different values for n: 2 (a-b), 5 (c-d), 15 (e-f). We used $a = b = 0.85$ for all three curves. In the top row (a, c, e) the region of fading is shown between the white border indicating $s = 1$, and the red border indicating $s = 1 + \alpha$.

Then, for each pixel with normalized image coordinates (x, y), we evaluate Equation 3.9. If $s < 1$ we leave the opacity of the pixel at 1. Otherwise, if $s \geq 1$, we perform an interpolation to determine the opacity of the pixel. The interpolation is done from $s = 1$ indicating an opacity of 1, to $s = 1 + \alpha$, where α is some small value determining the length of the interpolated section, indicating an opacity of 0. We use the smoothstep function for the interpolation. The smoothstep function $S(x)$ is defined as:

$$S(x) = 3x^2 - 2x^3 \tag{3.10}$$

for $0 \leq x \leq 1$. If $x < 0$ then $S(x) = 0$ and if $x > 1$ then $S(x) = 1$. To apply this function, we must first normalize the value s from the range $[1, 1+\alpha]$ to the range $[0, 1]$ which is done with the following equation.

$$s' = \frac{s-1}{\alpha} \tag{3.11}$$

We can now compute $S(s')$ following Equation 3.10. The smoothstep function interpolates between 0 and 1, but we want the opposite, i.e., the closer a pixel is to the border, the lower s is, the higher the opacity should be. This can be achieved by setting the opacity of the pixel equal to $1 - S(s')$.

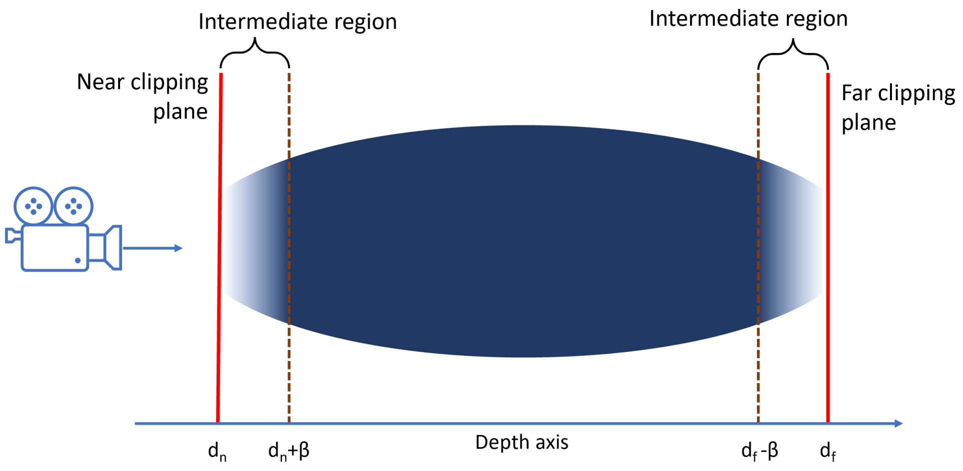


Figure 3.12: Schematic illustration of the soft depth fade. The near clipping plane is located at depth d_n from the camera. Between depth d_n and $d_n + \beta$, and similarly between $d_f - \beta$ and d_f, we perform a linear interpolation from opacity 0, transparent, to opacity 1, opaque. Without the soft depth fade, the opacity would be 0 outside the interval $[d_n, d_f]$ and 1 within.

In addition to this fading at the edges of scenes within the poster, we wanted to make objects fade in or out of the image in depth. Commonly, the clipping planes make hard cuts in the scene, i.e., everything that is between the clipping planes, and within the frustum, is completely visible, and everything that is outside is completely invisible. We wanted to create an effect of a soft cut. To achieve this we added an intermediate region, within the view frustum close to the clipping plane. This is the region where the opacity of objects fades towards zero, and becomes zero at the clipping planes, as shown in Figure 3.12. Depth is used to determine the opacity. If the depth is smaller than or equal to the depth of the near clipping plane, the opacity is zero, i.e., the object, or section of object, is completely invisible. Similarly, if the depth is greater than or equal to the depth of the far clipping plane, the opacity is also zero. To create the fading effect, we interpolate from 0 to 1 in opacity in the intermediate region with depths in the interval $[d_n, d_n + \beta]$, where d_n is the depth of the near clipping plane and β is the length of the intermediate region. To compute the near depth fade of a point with depth d which is within this region, we first normalize the depth from the range $[d_n, d_n + \beta]$ to the range $[0, 1]$ as follows:

$$d' = \frac{d - d_n}{\beta} \tag{3.12}$$

We then apply Equation 3.10 and compute $S(d')$ which gives us the opacity of the point. A similar approach is used for the far depth fade, although the interval of depths for the transition region is $[d_f - \beta, d_f]$ where d_f is the depth of the far clipping plane, and the result of the interpolation would have to be transformed by computing $1 - S(d')$.

Chapter 4

Implementation

We developed a prototype of TimeBender using the Unreal Engine (UE). In this chapter we explain the implementation of this prototype and how the UE Application Programming Interface (API) was utilised to realise the implementation. First, we give an overview over UE and the structure of the project. Second, the implementation of the three pipeline steps is described. Then, we explain how the focus object is modelled as a spline using the UE API, the implementation of soft fading without the use of translucent materials, and finally, the saving and loading system.

4.1 Unreal Engine

As mentioned above, the prototype was developed using UE [35]. UE is a well known game engine created by Epic Games to render real-time 3D graphics. It is used to develop games, create virtual sets and special effects for film and television, and prototype architectural and vehicle designs, among other application. An example showcasing the rendering power of UE is the show *The Mandalorian*, where the whole show was filmed using virtual sets created in and rendered by UE. UE is also used for academic purposes, for example in the field of computer vision where realistic virtual environments can be used to train robots for real world tasks[63].

UE is written in C++ and contains more than 2 million lines of code. The current latest version is 5.0.0, which was released during the course of this thesis. To avoid porting issues, we used the same version of the engine throughout the project, which was version 4.27.0, the newest version at the start of the project. When developing in UE, we have access to the full source code, although this mostly does not have to be considered in detail. In fact, it is possible to create games, or other projects, without touching C++ code. From the UE editor, developers can use a custom visual scripting language, i.e., a programming language in a graph-like visual form with nodes as code blocks and links as data or control flow. The visual scripting language is compiled to C++ code by the engine. However, the recommended method is to combine writing C++ code and visual scripting since the C++ classes have access to more engine functionality and creates more compact code, whereas the visual scripting language is less complicated and has shorter development iterations as the engine does not have to be rebuilt and restarted between coding updates.

UE was chosen for this project as it is a powerful rendering engine with a vast API and a helpful community of developers. By providing access to the source code it allows for flexibility, while the API and community resources such as example code, forums, and tutorials, prevent us from having to reinvent the wheel. UE also has a plugin system which makes sharing code between users of the engine convenient. There are plugins developed by Epic Games which are by default available in the engine, but have to be activated. Plugins developed by the community can be shared, for example, as Github repositories. To use such a plugin, one has to download the code and place it in the plugin folder of the UE project, as shown in Figure 4.1. When starting up the engine with this project, the plugin will be available to enable and use. Because of this convenient code sharing system, we implemented our prototype as a plugin.

4.1.1 Project Overview

Although there are many benefits to using a powerful and versatile engine such as UE, it lays constraints on the structure of the project. Notably, there is a separation between the editor and runtime modes. The editor mode is where the development outside of C++ coding happens, e.g., the 3D setup of levels, visual scripting, etc. The runtime mode is what the end user of the project would see, e.g., a finished game. The only Graphical User Interface (GUI) in the runtime mode is that which has been created by the developers, whereas in the editor mode we have the engine GUI created by Epic Games. This separation also had implication for the development of our framework.

The plugin that was developed as a prototype is split into two modules, one for the editor and one for the runtime. In the editor module, changes were made to the GUI of the UE editor. We added an additional GUI widget for the interactions needed when creating a poster, shown in Figure 4.2. The editor module also takes care of the logic when adding new scenes, and when snapping the view to a scene camera.

The runtime module handles what happens when pressing play in the editor, i.e., when running the project in the engine. It contains the Game Mode, a class deriving from the *AGameModeBase* class which sets up the runtime GUI and the pawn, i.e., the object that controls the interactions of the user in runtime. This module also contains the pawn class, deriving from the *APawn* UE class. This class handles all interaction logic, and is the largest and most important class in the module. The classes responsible for the runtime GUI, in addition to the scene and spline classes deriving from the UE *AActor* class, are also part of the runtime module.

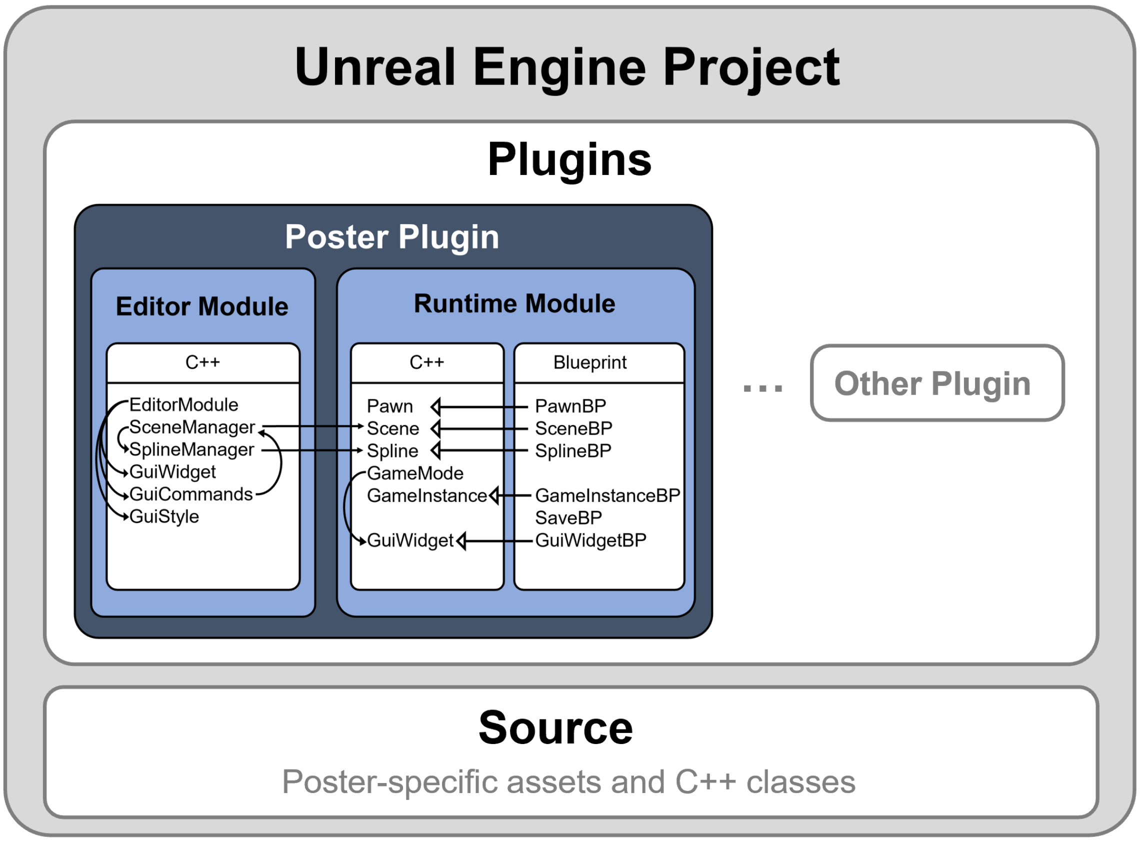


Figure 4.1: An overview of the project structure. Within the project, there are two separate folders holding code, the plugin folder and the source folder. Within the plugin folder is the Poster Plugin, but depending on the authors there could be other plugins as well. The source folder holds poster-specific code and assets, i.e., code and assets that are not part of the framework, but added by the author to create a poster.

In addition to the C++ classes, we add functionality with visual scripting in the editor. Benefits of this is that the development iterations are a lot quicker as the editor does not need to be rebuilt and started for each code edit. A drawback is that visual scripting quickly turns into very large graphs with nodes, i.e., code snippets, and edges, i.e., data passing between nodes or the flow of the program. Following the logic of the program, for example, to find bugs or change functionality, quickly becomes difficult.

4.2 Main Pipeline

In this section we detail the implementation of the main pipeline described in Section 3.1. In this implementation of the framework, the pipeline is not strictly linear. This is because the local layout step happens in the editor mode, whereas the two other steps happen in runtime mode. This means that the author can stop the pipeline at any time during the global or detailed layout steps and go back to the local layout, by simply exiting the runtime mode.

4.2.1 Local Layout

The local layout is separated from the two other steps by the fact that it takes place solely in the editor of UE. The implementation for rendering a 3D scene, adding models and lighting, and interacting with these is provided by the engine. We needed to extend this implementation such that the author can set up multiple scenes within one level, and snap the viewport to the camera of a given scene. To add this functionality to the UE editor, we created a custom widget and attached it to the editor GUI. This widget is an instance of a class deriving from the *SCompoundWidget* class and based upon the Slate UI framework which is integrated in the engine. The widget can be seen in Figure 4.2.

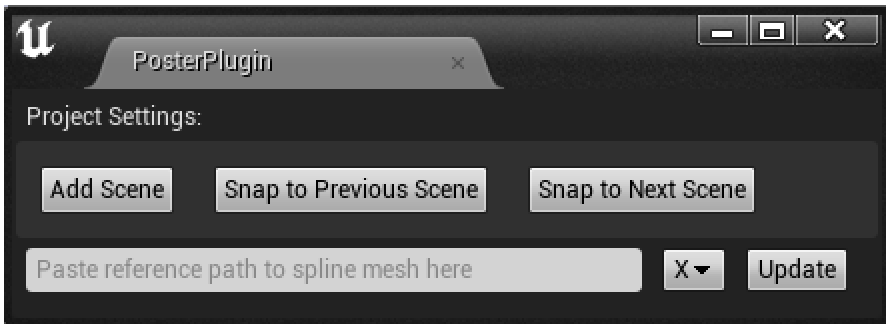


Figure 4.2: The GUI widget added in the UE editor to enable the additional interactions necessary for creating a poster.

In the first horizontal line of the widget there are three buttons: one to add a new scene, one to snap the view to the camera of the previous scene, i.e., the scene with index one smaller than the current scene, and one to snap the view to the next scene camera. When snapping the view to a scene camera, the author can see the scene from the same perspective as it will be shown in the final poster. In the second horizontal line, there is a text field where the user can paste or write the reference to the mesh that should be used as the focus object spline. To access the reference to a mesh, the author must right click on the relevant mesh, and select copy reference in the menu that appears. This is explained to the user in the tooltip of the text field, i.e., an additional explanation text that appears when hovering the mouse pointer over the text field. Next to this is a drop down menu where the user can select the forward axis of the chosen mesh, i.e., the axis aligned with the spline, and a button to update the spline mesh.

Snapping the viewport of the editor to a scene camera is done by accessing a reference to the editor via the *GEditor* variable, getting the viewport client from this reference, and setting the view location and rotation of this client to be equal to the location and rotation of the scene camera in question. The author can change the view of the viewport by the regular interactions, i.e., scrolling, clicking and dragging, but can always snap the view back to a camera position to see how the scene will be portrayed in the final poster.

Adding scenes is more complicated, and we will start by explaining what a scene is in this implementation, then how the scenes are managed and created. First, the scenes are modelled in two layers, a C++ class deriving from the *AActor* class, and a blueprint class deriving from the C++ class, as shown in the project overview in Figure 4.1. The reason for these two layers is to facilitate development based on the UE structure. In the blueprint class, we can join different components into more complex structures, and quickly add functionalities to these components with the visual scripting system. The blueprint of a scene consists of two components, a *SceneCaptureComponent2D (SCC)* and a *StaticMeshComponent* in the shape of a plane. The *SCC* is equivalent to a camera, but it writes to a render target that can be used as a material, instead of being viewed directly in the viewport. The plane acts as a canvas by having a dynamic material that is written to by the *SCC*. The dynamic material is updated every frame, which enables animation in the scenes of the final poster. The *SCC* has a parameter called *Primitive Render Mode* which can be used to change what is rendered. We set this to *Use ShowOnly List* such that only the models that are in the *ShowOnly list*, i.e., that are specified to be a part of this scene, are rendered. All actors that are tagged with Scene$< n >$, $< n >$ being replaced by the scene index, are added to the *ShowOnly list* when entering runtime mode.

In the level hierarchy of the UE editor, all scenes are added to a folder called Scenes, and named SceneBP$< n >$, $< n >$ being replaced by the scene index. This is handled by the *SceneManager* class which is part of the editor module of our plugin. The *Scene-Manager* is a singleton class, i.e., there can only be one instance of the class. This is realized by having a static member variable that is a pointer to an instance of the class, and a static member function to access an instance on the class. This function checks if the member variable has been populated, and if it has, returns the value that is stored in the variable. If not, it creates a new instance and stores a reference to it in the member variable before returning this reference. We have to make sure that there is only one instance of the *SceneManager* class since it is responsible for managing the scenes, e.g., by making sure that no scenes have the same index.

When creating a new instance of the *SceneManager* class, we first access and store references to the scene class and the world, i.e., the top-level component of the 3D level. Next, we iterate through all actors of the scene class within the world to reset the indices, e.g., to handle a case where a scene has been deleted, and count the number of scenes in the world. This number is stored for use when new scenes are added to the poster. When a new scene should be added, the *SceneManager* instance uses its world reference to spawn a new actor of the scene blueprint class into the world. It is placed at a location based on its scene index, i.e., the number of scenes before this scene was spawned.

4.2.2 Global Layout

When pressing "Play" in the editor, the user enters the runtime mode, and consequently the global layout step of the pipeline. This action also triggers the *BeginPlay* events in all actors, as well as spawning the pawn, i.e., the object responsible for capturing and handling user input. The pawn is a very important class in runtime mode, for example, in a game project, an instance of the pawn class would be the object controlled by the player or an artificial intelligence in the game. In this project, the pawn is the class that controls and handles all user input, and sends instructions to objects of other classes, e.g., for the scenes to move.

When entering this step of the pipeline, the canvases from each scene are placed in a horizontal line in front of the main camera that renders the final poster. The position of each scene in the main camera space, i.e., the coordinate system centered and aligned with this camera, is calculated according to the method described in Section 3.1.2. The canvases are static meshes in the shape of planes, with a dynamic material written to by the scene camera. In this way we stitch together the views of the scene cameras and the main camera of the poster. In the global layout step, the scenes are shown with an outline to better indicate their scale and location within the poster. The outline is created in the material blueprint of the canvas, which is a shader program defining how the canvas is rendered. The texture coordinates of the current pixel, i.e., the canvas is parameterized in two dimensions such that every point on the canvas is mapped to a value in a 2D texture, are accessed by the *TexCoord* node. The texture coordinates form a 2D vector of floats with values between 0 and 1. These values are mapped to the range $[-1, 1]$, then the 2D vector is evaluated with the superellipse equation described in Section 3.3 with $a = b = 0.9$ and $n = 5$, both of these values were chosen empirically and could be changed to fit the needs of the poster. If the 2D vector is outside the superellipse curve, i.e., the result of the equation is larger than 1, we do a linear interpolation between the color of the texture and the color of the border with the alpha value of the interpolation equal to the result of the superellipse equation minus 1. The color of the texture corresponds to the rendering of the scene camera.

In the global layout step, the author decides the layout of the scenes within the poster, similar to comics-based visualizations, by changing the image space location as well as the size, i.e., the depth in camera space, of the scenes. The image space location of a scene is changed by clicking on the scene with the mouse pointer and dragging it to a new location. This is handled in the pawn blueprint class. We added a *LeftMouseButton* event which has two sub-events: *pressed*, which is triggered when the left mouse button is pressed, and *released*, which is triggered when it is released. In the *pressed* sub-event, we use a *LineTraceByChannel* node to check if a line segment starting from the mouse pointer and going into the world hits a scene. This line segment is calculated by using the *ConvertMouseLocationToWorldSpace* node, which takes the player controller as input, and as output it returns the world space location of the mouse pointer as well as the direction it is pointing in world space, i.e., the direction pointing towards what is "behind" the mouse pointer on the screen. The line segment is then defined as starting with the world space location of the mouse pointer, and ending at 10.000 times the world space direction added to the world space location. 10.000 as multiplier is chosen empirically and covers what can be seen on the screen, i.e., if something is further away than this it most likely will not be visible to the user. The *LineTraceByChannel* node returns the closest object that intersects with the defined line segment as a *Hit result* structure. From this structure we extract the *OutHitHitActor* and check if it is of the correct class type, i.e., a scene blueprint class. If this check is true, the pawn tells the hit scene that it has been clicked, by triggering a custom event. In this custom event, the scene sets a boolean member variable, indicating whether it is currently clicked, to true. In the *tick* event of the scenes, i.e., an event that is triggered every frame, this boolean member variable is checked. If it is set to true, the scene is moved such that it has the same image space location as the mouse pointer, without changing the depth of the scene in camera space. If it is set to be false, the scene does not move. In the *released* sub-event of the *LeftMouseButton* event, the pawn class triggers all scenes to set their boolean member variable indicating whether they are currently clicked back to false.

The depth of the scenes are controlled by a slider which is part of the runtime GUI. The runtime GUI is created in a widget blueprint, deriving from the *UUserWidget* class. A widget blueprint is a combination of a GUI designer and visual scripting, which can create both the look of the interface as well as the behaviour. A slider is one of the default components and its behaviour is mostly predefined and parameterized. We can, for example, change the design, the minimum and maximum values, the default value, etc. It also has events that automatically trigger when interacting with it, e.g., the *OnValueChanged* event that triggers when a user clicks and drags the slider such that the value is changed. When this event is triggered, the GUI broadcasts it with an event dispatcher that is listened to by the pawn class. In the event dispatch are two variables, the new slider value and the index of the currently selected scene which is decided by a spin box component in the GUI. Based on the new slider value, the new scene depth is calculated by the pawn. The slider value is within the range $[-1, 1]$, where -1 corresponds to the maximum depth, i.e., the minimum size, and 1 corresponds to the minimum depth, i.e., the maximum size of a scene. A slider value of 0 corresponds to the baseline depth, i.e., the depth the scenes are placed at in the default layout. Since this baseline is shifted depending on the number of scenes in the poster, the ranges $[min, baseline]$ and $[baseline, max]$ are not necessarily of equal size. Because of this, the mapping from a slider value to a scene depth must be done separately depending on if the slider value is in the range $[-1, 0[$ or $[0, 1]$. If it is in the first range, the slider value, s, can be mapped to the scene depth, d, by the following equation:

$$d = (s + 1)(baseline - min) + min \tag{4.1}$$

Otherwise, the slider value, s, will be within the second range, $[0, 1]$, and can be mapped to the scene depth by the following equation:

$$d = s(max - baseline) + baseline \tag{4.2}$$

When the new scene depth has been calculated by the pawn, it locates the relevant scene in the array of scene references that it stores. The scene class has a function to move its canvas to a depth value given as input to the function, without changing the image space position of the center point of the scene. The implementation of this function follows the method described in Section 3.1.2. This function is called by the pawn with the newly calculated depth on the selected scene, and the canvas is moved accordingly.

4.2.3 Detailed Layout

To move on to the detailed layout step, the author must click a button in the GUI. When this is pressed, the scene locations are locked, and the focus object spline becomes visible. This is done by setting the *Visible* parameter in the spline blueprint class to true. Each control point of the spline is moved to the image space location they had during the local layout within the canvas of the related scene. The depth of the control point is set to the depth of the canvas of the scene added to the depth that the control point had in the scene camera space during the local layout step. This is crucial to achieve the preservation of the depth relationships from the local layout in the final poster.

Since the focus object is rendered by the main camera of the poster, whereas the models in the scenes are rendered as dynamic materials on canvases in the poster, we have to use the material blueprint of the scene to ensure that the depth relationship between the scene objects and the focus object are preserved. As described in Section 4.2.1, the scene camera is a *SCC* which has a parameter, *CaptureSource*, that can be set based on which information is needed in the material blueprint. We set it to *SceneColor in RGB, SceneDepth in A*, since we do not support translucency in the scenes we do not need the opacity in A. This 4D vector, RGBA, is sent to the material blueprint as a parameter, which means that we have access to the depth of the scene, as well as the color, per pixel. To simulate the depth relationship between the focus object and the scene objects in the final poster, we compare the depth passed to the material blueprint from the scene blueprint, with the depth buffer of the poster which is accessed by the *SceneTexture:SceneDepth* node. To avoid confusion since this value is also called SceneDepth, we rename the value retrieved by the *SceneTexture:SceneDepth* node to poster depth in the rest of the thesis. Before we compare the scene depth with the poster depth, we must subtract the depth value of the canvas from the poster depth since we want to compare the depths as if the focus object was part of the scene. It would be equivalent to instead add this depth value to the scene depth. After this we can compare the scene depth with the altered poster depth to figure out the correct depth relationship between the two. If the poster depth is smaller than the scene depth, the opacity of the scene at the current pixel is set to 0. This is because it is supposed to be behind the focus object, i.e., not visible. If the scene depth is smaller than the poster depth, the opacity value of the pixel is set to the determined value based on if it is on the border, if it shows only background, or if it shows a scene object.

The image space position of control points can be changed by clicking and dragging. This is implemented in a very similar way to clicking and dragging scenes, as described in the previous section. Therefore, it is not described in detail again. One new consideration when moving control points instead of scenes that is important to note is that if the author is using tangents approximated with central differences, as described in Section 3.1.3, moving one control point influences more than just the spline segments adjacent to the moved control point. The position of a control point affects the tangents of neighbouring control points, which in turn affects the spline segments adjacent to the neighbouring control points. When moving a control point, we must, therefore, recompute four spline segments: the two adjacent to the moved control point, as well as their neighbours on each side. When using the author-defined tangents, we only have to recompute the two adjacent spline segments of the moved control point.

Textual elements can be added by two buttons in the GUI, one for titles and one for annotations. These textual elements are also GUI components. In fact, they are created as their own widget blueprint, one for titles and one for annotation, inheriting from the *UserWidget* class. They both consist of a *EditableText (Multi-Line)*, a checkbox to determine whether or not to have a contrasting background, and a slider to determine the width of the text box. The reason why we have two widget blueprints is that UE does not allow for editing fonts during runtime. This means that the font must have a set size, color, emphasis, and justification before entering runtime mode. With this limitation, we decided to create two types of textual elements with preset fonts. For the title this is a large font size, black color, bold emphasis, and central justification, which is a common setup for a title text. For the annotations, we use a small font size, black color, no emphasis, and left justification. In both blueprints we added logic to hide the components, except the text, when the mouse pointer was not on the widget. This was done since these textual elements were part of the final poster, and the additional components, although adding valuable interactions, should not be. When clicking the button to add one of the textual element types, an event is triggered in the GUI. We use the *Construct* node with the relevant class, e.g., the title widget blueprint class if the title button was clicked. The newly spawned object is then added as a child to the main GUI widget. This places the new textual element in the top left corner of the screen, i.e., at coordinates $(0, 0)$ in image space. We wanted the textual elements to be moved with a click and drag interaction. To do this with GUI components required some additional classes. Firstly, we needed a blueprint class deriving from the *DragDropOperation* class. In the blueprint we added two additional variables, a 2D vector of floats to hold the offset of the dragging action, and a reference to a *UserWidget*, to hold a reference to the textual element that is being dragged. Then we needed a widget to act as the temporary dragged object. This was a simple version of the textual element widget blueprint, also deriving from *UserWidget*, and containing only the text component. In the widget blueprint of the two textual element types, we overrode the *OnDragDetected* function. In this function we created a temporary draggable text widget containing the same text as the textual element, and with the same width. Then we create an instance of the class inheriting from *DragDropOperation* with the temporary draggable widget as the *DefaultDragVisual* input, the position of the mouse pointer within the textual element widget as the offset input, and a reference to itself as the widget reference input. In the main GUI widget blueprint class, we overrode the *OnDrop* function. This function is called by the *DragDropOperation* when the user releases the mouse button. The *DragDropOperation* instance is an input to the *OnDrop* function, and since we stored a reference to the dragged textual element in this object, we can access it and change its location to the current location of the mouse pointer. The destruction and removal of the temporary draggable text is handled by the *DragDropOperation*.

4.3 Focus Object Modelling

As discussed in Section 3.2.2, we decided to use a spline based deformation approach for the focus object. Conveniently, UE has a class that is well adapted to this use, the *USplineComponent* class. This class creates spline objects that can be placed in the 3D space of the level. An arbitrary number of control points can be added to the spline. The position of the spline points and the direction and scale of the entering and leaving tangents can be adjusted by the user. In Figure 4.3 a UE spline object can be seen. The third control point is selected and can be moved around in 3D space. The white line segment intersecting the point defines the direction and scale of the entering and exiting tangent at this control point. By selecting the white square at one of the ends of the line segment, the tangent can be altered, although the tangents are constrained to stay colinear.

In UE, the splines are piecewise cubic Bézier splines, i.e., there is a cubic Bézier spline between each pair of neighbouring control points. Cubic Bézier splines are define by four

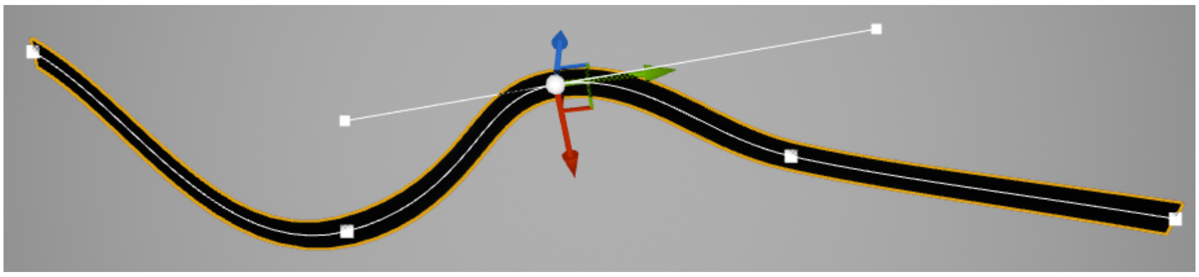


Figure 4.3: An example spline from the UE editor view. The thin white curve is the *USplineComponent*, whereas the black thick curve is the mesh deformed by the *USplineMeshComponent*. The third control point is selected, which is indicated by the glyph showing the direction of the three main axes. The line segment passing through the selected control point shows the entering and exiting tangents at this control point.

control points. In the UE implementation, we have the two control points that are the start and end points of the spline segment, and the two intermediate control points are derived from the original control points and the start and end tangents.

The *USplineComponent* class contains only a logical spline, not visual. This is because the component is often used for non-visual splines, for example, camera paths, animation of models, etc. When using the component for a spline that should be visible, we must add an instance of the *USplineMeshComponent* class for each spline segment. This component handles the deformation of a mesh along the spline segment. For each instance we must define the mesh to be used and the forward axis of this mesh, i.e., which axis to align with the spline. We must also specify the start and end location, the start and end rotation around the spline axis, as well as the start and end tangent. With this information, the component deforms the mesh along the spline, as can be seen in Figure 4.3.

4.4 Soft Fading Effects

Rendering translucent objects is an expensive process compared to rendering opaque objects. With opaque objects, the renderer only has to check which object is the closest to the camera per pixel, and render this. Whereas, with translucent objects, multiple objects can be visible per pixel, and to render the pixel a composition of the contributions from each visible object must be considered. Because this process is so expensive in terms of computational complexity, UE has made some simplifications when rendering transparent objects. In fact, there are multiple rendering passes in UE, wherein one renders all opaque objects, and one all translucent objects are. The opaque objects are rendered first, and their depths written into the depth buffer. Translucent materials are rendered back-to-front based on the depth of the component. Per pixel, the translucent material checks if there is an opaque object in front of the translucent object, and in this case it is not rendered. Since the back-to-front composition is not based on the depths of objects per pixel, but the depth of the object, the sorting of translucent objects is only correct if they are spread out sufficiently in space. Otherwise, or if there is an object consisting of multiple meshes joined together, sorting of translucent objects is wrong. An example is shown in Figure 4.4.

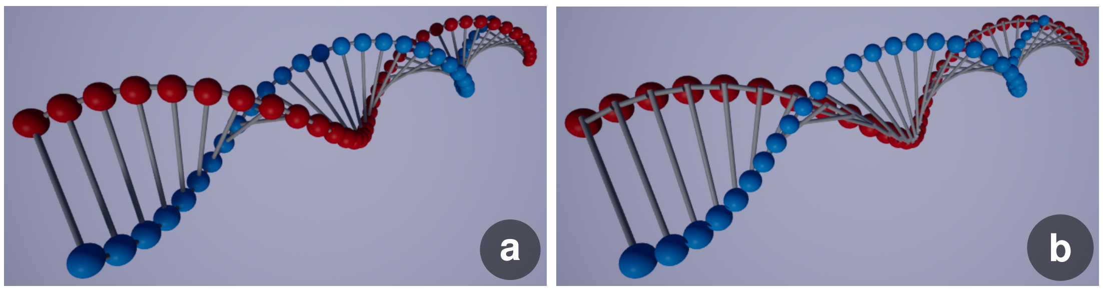


Figure 4.4: The figure shows a model of a DNA molecule rendered with an opaque material (a) which results in correct depth sorting of the component, and the same model with translucent materials (b) resulting in incorrect depth sorting. The spheres with the blue material are always rendered on top, the cylinders with the white material are rendered in the middle, and the spheres with the red material are rendered in the back.

This sorting issue and high performance cost are known limitations of using translucent materials in UE. A possible workaround is to change the blend mode of the material, i.e., how the material is blended with the background, to masked. In the masked blend mode pixels are either drawn as opaque or not drawn at all, i.e., parts of the object are completely transparent, and the rest is completely opaque. The masked blend mode allows for traditional screen door transparency by using using a masking material that, e.g., removes every other pixel in a checkerboard pattern. Screen door transparency creates visual artifacts and improvements have been proposed to this old technique. One of these improvements are called stochastic transparency [30] and basically uses the idea of screen door transparency on a sub-pixel level with a random screen door pattern, i.e., not a checkerboard. The sub-pixel level means that each pixel is split into multiple segments, i.e., the image is super-sampled. The opacity of an object is equal to the proportion of sub-pixel segments that are covered by the object. We implemented an approach inspired by this, but in addition to supersampling the image, we use temporal anti aliasing, i.e., averaging the pixel colors over the last few frames. Temporal anti aliasing is turned on by default in UE, and can be found under project settings → engine - rendering → default setting → anti-aliasing method.

To implement the approach we created a new material blueprint, i.e., a shader program

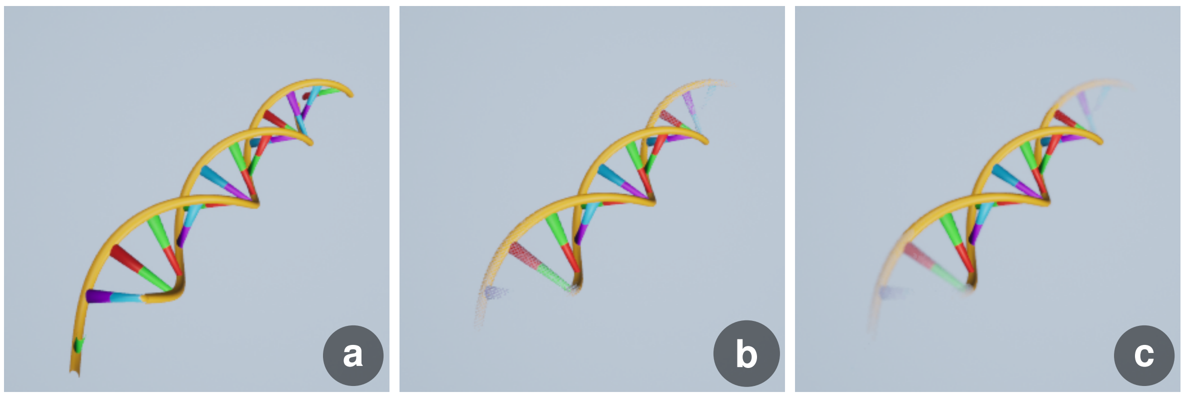


Figure 4.5: Comparison of a hard clipping plane (a), a soft fade clipping plane using the *DitherTemporalAA* node (b), and a soft fade clipping plane using the *DitherTemporalAA* node and applying a Gaussian blur to the faded region (c).

using visual scripting. The material uses the masked blend mode and the opacity mask, i.e., which pixels are drawn, is decided based on probability and randomness using the *DitherTemporalAA* node. This node takes two inputs, first, the alpha threshold which is the probability of a pixel being drawn, equivalent to the proportion of sub-pixel segments in stochastic transparency. Second, an input to indicate whether this opacity mask should be random or not. Based on these inputs, the *DitherTemporalAA* node creates a stipple pattern that is more or less dense depending on the alpha threshold. An alpha threshold of 1 indicates that all pixels should be drawn, and of 0 indicates that no pixels should be drawn. We determine this value by using a smooth step function with the depth of the pixel as input as described in Section 3.3. The depth value of the pixel is an input to the material blueprint and can be accessed through the *PixelDepth* node. An example can be seen in Figure 4.5 b.

Some noise remains when using this stochastic transparency technique, which is especially noticeable when the transparent object and the background have highly contrasting colors. To reduce the noise further, we apply Gaussian blurring in the intermediate regions of the near and far depth fade, as shown in Figure 4.5 c.

For fading the edges of the scenes in the final poster, we use regular alpha-blending transparency. We can do this because it is done in image space, i.e., there are no sorting issues of the objects within a scene, and the scenes will be the only transparent objects in the poster. We know that these will be sufficiently separated in space as placing scenes on top of each other in the poster does not make sense. We therefore do not have to worry about the sorting order of the scenes within the poster.

4.5 Saving and Loading

For the convenience of the authors, including ourselves, adding saving and loading functionality to TimeBender was important. UE has a well integrated system for saving and loading which is crucial for a game engine. The highest level of saving in UE is the project. This includes an instance of the engine code, in addition to any code added by the developers. By running this project, either through the Epic Games Launcher, or by running the code through an IDE, e.g., Visual Studio, an instance of the engine is started. Within this project, there can be one or more levels, which are 3D scenes. Assets, e.g., 3D models, materials, etc., are loaded into the project and can be placed in the levels. This is saved automatically by UE when pressing *ctrl+s*, or selecting save in the file menu. In our prototype implementation we use a project as an instance of our framework which can produce multiple posters. Each level corresponds to one poster, and models can be shared between posters easily since they are tied to the project and not the level, i.e., the poster. We do not have to add any functionality to save the projects or the levels since this is handled by UE. This corresponds to saving in the local layout step of the main pipeline of the framework, including the models that are part of a scene, their 3D position, rotation and scale, animation in the case of an animated model, as well as the 3D positions of the focus object spline control points, and their tangents' orientations and scales.

A lower level of saving in UE is saving instances of running a project, e.g., saving the progress of a player in a game. In the case of TimeBender, saving an instance means to save a poster that uses the same local layout but has a different global and detailed layout. Allowing for saving instances of a poster is useful, for example, when the author knows the narrative they want to present, but is unsure about the final look of the poster. They can then use the same local layout to create posters with different layouts and annotation to compare their impact. Saving instances also allows for editing a poster after creation, e.g., to fix mistakes or incorporate feedback. There is a good system for saving instances of running a project in the API, but it requires an implementation for the project in question. To save an instance of the project, a class inheriting from the UE *SaveGame* class must be created. In this class we define the data that is saved per instance. But first, we must think about what information needs to be saved.

The information to save depends on which step in the pipeline the author has reached. If the author is in the local layout step, it does not make sense to consider saving instances because all information in this step is saved when saving the 3D scenes. If the author has reached the global layout step, saving an instance of the poster means saving the position of the scenes in the camera space of the main poster camera. This includes the 2D image position as well as the depth, and with this also the size, of the scene, creating a 3D vector which must be stored per scene. If the author has reached the final step, the detailed layout, the information from the global layout step must be saved, in addition to the new information provided in this step. This includes the 2D screen position of the control points related to each scene, the choice of tangent, i.e., author-defined or central differences approximated, the scale of the tangents for each scene, and finally the textual elements, both titles and annotations, which are not related to scenes. For the textual elements we must store the textual content, the 2D screen position, the width of the text box, and if there should be a contrasting background or not.

Related data can be grouped into *structures* to make the saving neater. We define two structures, one for holding scene data and one for holding text data. In the scene structure we save the index of the scene as an integer, the position of the scene in world space as a 3D vector of floats, and the world space position of the spline control point related to this scene also as a 3D vector of floats. In the text structure we save a boolean value which indicates whether the text is a title or not, a *Text*, which is an UE wrapper for a string, containing the textual content, the screen position of the text as a 2D vector of floats, and the width of the text box as a float. In our *SaveGame* derived class we must specify the types of data that is saved per instance. We, therefore, add an array of scene structures and an array of text structures. In addition, we add an integer value that tells us which step of the pipeline the instance was saved from, a boolean that indicates which type of tangent was used in the instance, and two arrays of floats holding the tangent scales for the two different types of tangents. If the poster instance is saved from the global layout step, only the relevant data is populated and the rest is left empty. The directions of the author-defined tangents are saved in the 3D scene, i.e., when saving the local layout.

In a class inheriting from the UE *GameInstance* class, the logic for saving and loading is handled. While in either the global or detailed layout steps, the author can choose to save an instance of the poster. This is done in a simple GUI with a text input field for the name of the saved file, and a button to save the instance with the written name, as shown in Figure 4.6. The name will be prefixed with the name of the poster, i.e., the level, to ensure that the saved instances are tied to the correct poster. An object of the class derived from *SaveGame* is created and the necessary data, as specified above, is extracted and placed into the appropriate parameters in the created object. Finally the

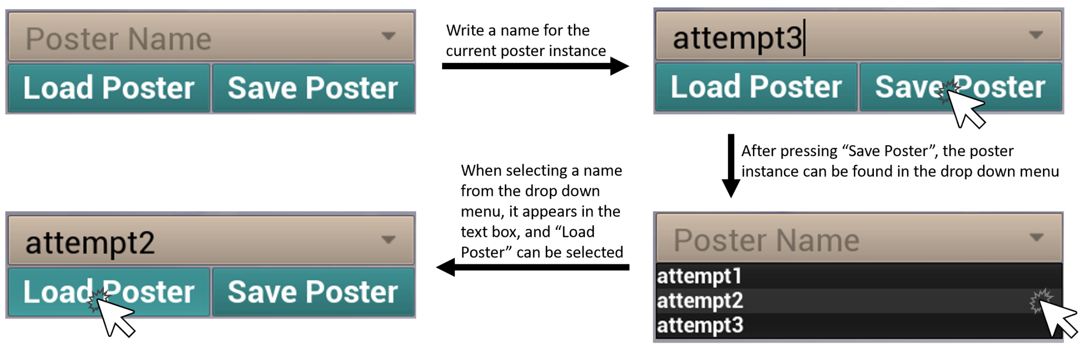

Poster Name
Load Poster
Save Poster
Write a name for the current poster instance
attempt3
Load Poster
Save Poster
After pressing "Save Poster", the poster instance can be found in the drop down menu
Poster Name
attempt1
attempt2
attempt3
When selecting a name from the drop down menu, it appears in the text box, and "Load Poster" can be selected
attempt2
Load Poster
Save Poster

Figure 4.6: An example of a saving and loading work flow. The author first saves the current poster instance as "Attempt3", before loading a previous attempt called "Attempt2".

SaveGame derived object is saved to a slot, i.e., a file, with the name specified by the user and prefixed by the poster name.

To make the loading of previously saved instances of a poster easier for authors using our framework, we created a drop down GUI element listing all saved instances from the current poster, shown in the lower right of Figure 4.6. To do this we enabled the *Blueprint File Utilities* plugin, developed by Epic Games, which allows us to list all files in a directory. The *ProjectSavedDir* function from this plugin returns the absolute path to the saving directory of the project. Slots, containing *SaveGame* objects, are located in a sub-directory called SaveGames. Listing all the files in the SaveGames sub-directory, and choosing only the ones that are prefixed by the current level name, gives us a list over valid saved instances of the current poster. These are listed in the drop down menu and can be selected by the author to be loaded. When loaded, the data is extracted from the *SaveGame* derived object, and passed to the relevant classes and objects.

Chapter 5

Results

5.1 Performance

To have an interactive authoring framework, it must run with interactive frame rates which we consider to be at least 20 Frames per Second (FPS). The posters can contain different amounts of scenes, and these scenes can contain different amounts of models with different complexities, i.e., number of vertices or polygons within the mesh of the model. For this reason we can only provide example frame rates for posters we have created, but this may vary with the complexity of the poster. The tests were run with UE version 4.27.0 installed on a desktop computer connected to a screen with resolution 1920x1080. The computer was running Windows 10 as operating system, with an Intel(R) Core(TM) i7-9700K CPU and a NVIDIA GeForce RTX 2080 SUPER graphics card. The results were as follows:

Poster	FPS in editor	FPS in runtime
Empty poster	120	120
The Genetic Journey	31	28
7 Wonders in 13 Days	22	19

Table 5.1: Approximate FPS values for posters with different numbers of scenes and models. The empty poster is added to provide a baseline for the rendering of the framework.

As can be seen in Table 5.1, our framework achieves interactive frame rates for the posters we have created in editor and runtime mode, except for the poster *7 Wonders in 13 Days* which has an approximate average FPS of 19 during runtime. This is quite close to the boundary of what we consider to be interactive, and the reason for this lower FPS is that this poster contains seven scenes with high resolution models, i.e., models with a high number of vertices. The frame rate could be improved by simplifying the models, i.e., removing vertices, but this would also remove details from the poster.

5.2 Usage Examples

In this section we present two usage examples created with the prototype implementation of our framework. Both of the usage examples contain animated models which cannot be captured in a written thesis. We provide two snapshots to give an impression, while the complete animation can be seen in the uploaded videos [2, 1].

5.2.1 The Genetic Journey

This first usage example is a simplified recreation of our inspiration visualization by Jennifer Fairman [69]. It tells the story of genetic research from a microscopic to a macroscopic level with four scenes and can be seen in Figure 5.1. The first scene depicts a chromosome that is the structure DNA coils up into on a microscopic level. The second scene contains a cell with different organelles, e.g., the nucleus shown as a split yellow sphere where the DNA is passing through. Within the cell, DNA is kept in the nucleus and dictates the behaviour of the cell based on the activated genes. The third scene shows a black human silhouette and a detailed heart. The heart is moving slightly towards to camera and back again to give the illusion of a beating heart. This scene illustrates how DNA can build different organs, e.g., the heart, by determining the behaviour of cells. The last scene shows a globe which is rotating such that there is not only one country or continent in focus. This poster is in a 3:1 format, i.e., the width is three times the height. This was chosen to imitate the inspiration poster by Fairman.

The poster includes the following models: a DNA strand [62] as the focus object, a chromosome in the first scene created in Blender [8], a cell [44] in the second scene, a human silhouette [27] and a heart [67] in the third scene, and a model of the earth [66] in the final scene.

5.2.2 7 Wonders in 13 Days

The second usage example was inspired by a travel blogger called Megan Sullivan [70]. She made a journey around the world, visiting the so-called 7 modern wonders of the world in 13 days. We used her journey as a story to frame historical facts about these important monuments. Each scene contains a 3D model of the monument it depicts, except the fifth scene showing the ancient city of Petra, where a textured plane was used in the place of a 3D model. In this poster, the scene models are not animated, whereas the focus object is. The focus object combines the visual connection metaphor of the red path or ribbon with an animated 3D model of an airplane that travels along this path. As can be seen in Figure 5.2, the plane is close to the Colosseum in the first image (a), whereas in the second image (b), the plane is about to leave the poster after the Great Wall of China.

The poster includes the following models: the temple of Kukulcan [57], Machu Picchu [64], Christ the Redeemer, which was an open source model, the Colosseum [5], a plane object textured with an image of Petra [28], Taj Mahal [22], and the Great Wall of China [53]. The focus object was created in Blender [8].

5.3 Expert Feedback

To gain feedback on the potential utility of our approach, we performed an expert interview with a traditional illustrator. She was not involved in the project and did not have any prior knowledge about the framework. The interview contained two parts to get feedback on the general concept of the framework, and the results created using the framework. First, the concept of the project was introduced and explained, and we showed her the inspirational example by Jennifer Fairman. After describing the creation pipeline, we asked the following questions:

1. What is your opinion on the separation of these steps (i.e., the local layout, global layout, and detailed layout)?
2. What is your opinion on the order of the steps?
3. How does this pipeline compare to the workflow of an illustrator? Do you, for example, rather use a linear workflow than a circular, i.e., iterative workflow, or vice versa.

We were given the following answers, transcribed as a paraphrased summary:

1. This separation is good and roughly follows the working steps of an illustrator. When starting a new illustration project I first have to think about how many stages (i.e., scenes) there are and what they should contain, in addition to the relative importance of each stage, for example, which should be the main focus and thus take up the most space.
2. The ordering of the steps is not so clean in an illustration project, usually there is an iteration between the local and global steps. The global step, i.e., how much space each scene has which is also influenced by the format of the poster, can influence the local step, for example, if the poster is in a portrait style, each scene would have more vertical space so models would be oriented accordingly.
3. The workflow is rather circular, i.e., iterative, between the local and global step as they influence each other, whereas between these two steps and the detailed step the workflow is linear. Textual elements such as titles and annotations, in addition to other details, are only added when the local and global layout steps are complete.

Based on these answers we know that our separation of steps in our pipeline is meaningful, although having a purely linear pipeline could be changed to a partially circular one, i.e., add iteration possibilities between the global and the local layout steps. It was also an interesting consideration that the global layout, and notably the poster format, i.e., aspect ratio, can influence the local layout.

In the next step of the interview we presented the two usage examples from Section 5.2. The posters were presented one by one on a screen with resolution 3000x2000. Each poster was shown both as a static image and as a video to capture the animation of the models. The expert then had time to examine them before we asked the following questions:

1. What are your initial thoughts when seeing these visualizations?
2. How would you describe the narrative flow in the posters?
3. From an illustrator stand point, what are good features of the visualizations?
4. What are bad features? How could the posters be improved?
5. What are use cases where you could see a poster of this type being efficient?

6. How would you, as illustrator, proceed to create a similar visualization in a standard tool such as Photoshop [4]?

We were given the following answers:

1. The visualizations look cohesive and it is easy to make out the narrative. The titles stand out clearly and provide context to the poster, i.e., guides the expectation of what one will see. The organic nature of the ribbon structure (i.e., the focus object), as well as the fading effects at the start and end, are pleasing to the eye and commonly used techniques in illustration.

2. The narrative stands out, but it assumes a ”western” audience, i.e., the scenes are laid out from left to right. For the *7 Wonder in 13 Days* poster, the animated airplane shows the direction of the narrative flow which makes it more accessible to other reading cultures that, for example, read from right to left. For the *The Genetic Journey* poster, the direction of the narrative flow could be encoded in, for example, a sequence in the animations, i.e., that the first scene is animated first, then the second scene starts its animation, and so on.

3. Based on illustration best practices, good features of the posters are that the hierarchy of the information is clear, i.e., it is clear what text represents the title, and which textual elements are related to which scenes. In addition, the coloring is good as the saturation of the focus object stands out and draws attention to the most salient feature. The animations add interest and guides the audience through the narrative.

4. Specifically for the *The Genetic Journey* poster, I would enhance the lighting, the poster is too dark. In addition, the black human silhouette draws the focus of the audience as it contrasts to the surroundings, and the black color could lead to a misinterpretation of the person being dead. There should also be an indication of the direction of the narrative flow as this could be interpreted differently depending on the culture of the audience.

5. One could, for example, use a neuron as a focus object since the neuron cell has an elongated structure called the axon. One could create a poster to explain a neural disease or how the signals from the brain travel through the body. Another potential usage example could be to use a blood vessel as the focus object, and set up scenes explaining the path of blood through an organ, or the progression of a disease through the blood.

6. It would take a long time and require much manual work. I would set up the individual scenes in a 3D modelling software such as Blender [8], and make sure that the scenes had a similar lighting and camera setup. Then, I would stitch them together in After Effects [3] and add the focus object.

Some conclusions that we draw from the answers to the second part of the interview are that the results created with our prototype implementation are meaningful and cohesive visualizations although we need to be mindful about how the direction of the narrative flow is encoded. Once the direction is known, the order of the scenes in the narrative is clear, but adding a visual cue to the direction is important, such as we did in the *7 Wonders in 13 Days* poster. The animation is an illustrative strength of the posters as it adds interest and guides focus, which indicates that this type of poster is best suited to be presented digitally.

We did not ask the expert to use our prototype to create a poster as the goal of this interview was not to evaluate the usability of the prototype, but rather the meaningfulness of the framework and example results from the perspective of an illustrator.

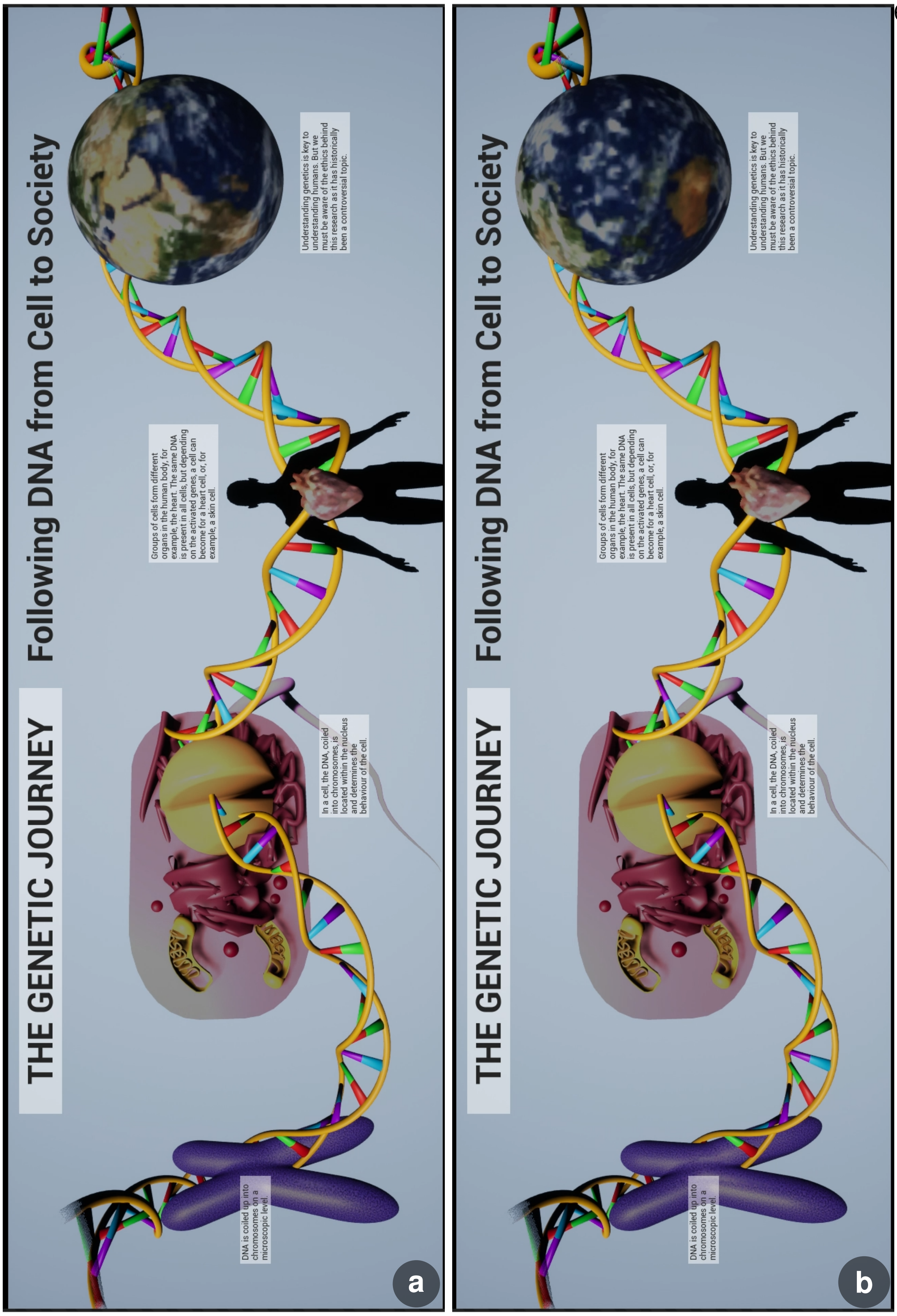
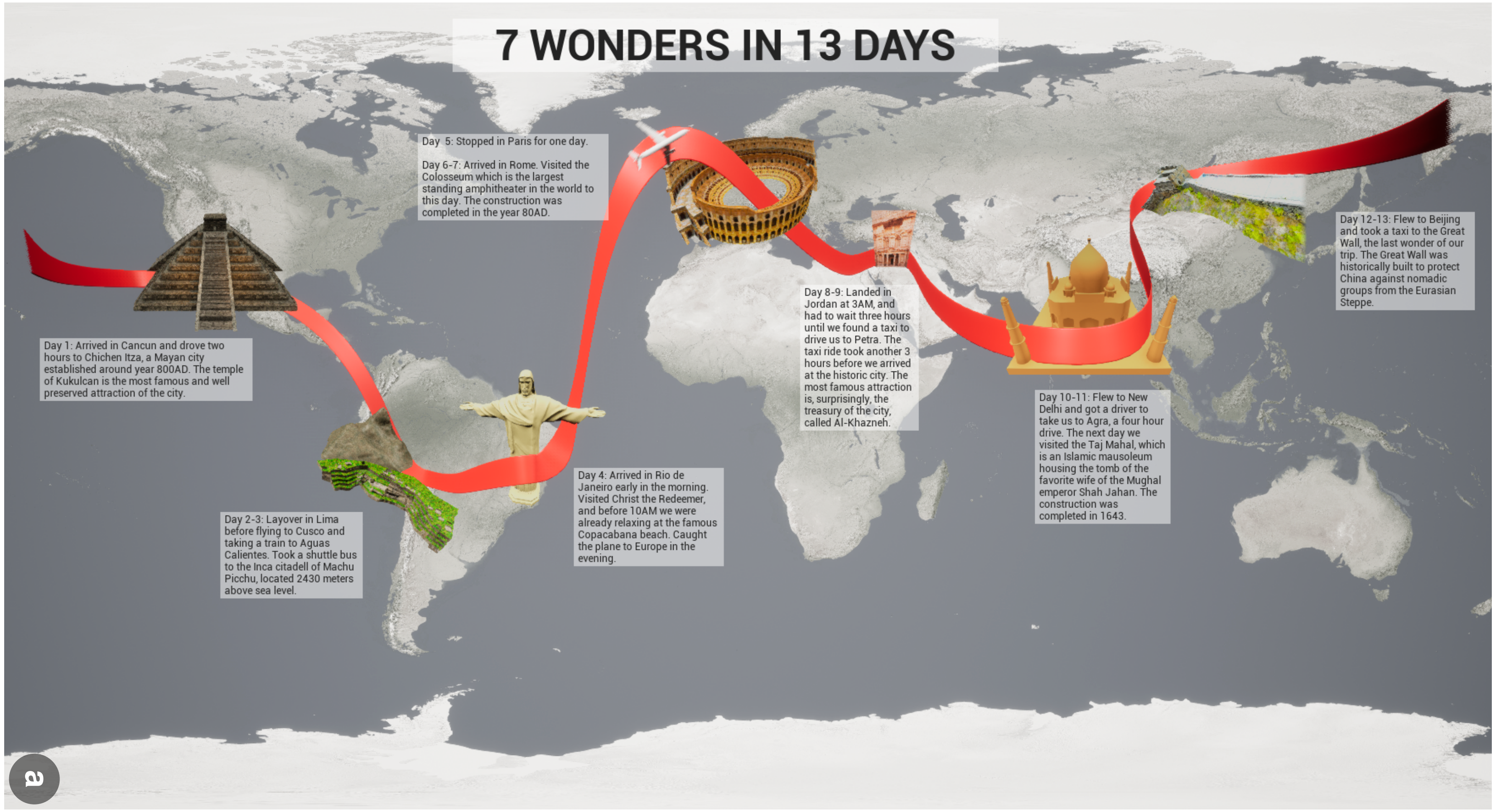

THE GENETIC JOURNEY
Following DNA from Cell to Society
DNA is coiled up into chromosomes on a microscopic level.
In a cell, the DNA, coiled into chromosomes, is located within the nucleus and determines the behaviour of the cell.
Groups of cells form different organs in the human body, for example, the heart. The same DNA is present in all cells, but depending on the activated genes, a cell can become for a heart cell, or, for example, a skin cell.
Understanding genetics is key to understanding humans. But we must be aware of the ethics behind this research as it has historically been a controversial topic.
a
b

Figure 5.1: Two snapshots of the animated poster, *The Genetic Journey*. In the first two scenes, one can see that the lighting changes between the images. In the third scene, the heart model is smaller, i.e., further away from the camera in the top image (a), and larger in the bottom image (b). The earth model is spinning and shows parts of Africa and the Middle East in the top image (a), whereas in the bottom image (b) it shows Australia and the Pacific Ocean.



7 WONDERS IN 13 DAYS

Day 1: Arrived in Cancun and drove two hours to Chichen Itza, a Mayan city established around year 800AD. The temple of Kukulcan is the most famous and well preserved attraction of the city.

Day 2-3: Layover in Lima before flying to Cusco and taking a train to Aguas Calientes. Took a shuttle bus to the Inca citadell of Machu Picchu, located 2430 meters above sea level.

Day 4: Arrived in Rio de Janeiro early in the morning. Visited Christ the Redeemer, and before 10AM we were already relaxing at the famous Copacabana beach. Caught the plane to Europe in the evening.

Day 5: Stopped in Paris for one day.

Day 6-7: Arrived in Rome. Visited the Colosseum which is the largest standing amphitheater in the world to this day. The construction was completed in the year 80AD.

Day 8-9: Landed in Jordan at 3AM, and had to wait three hours until we found a taxi to drive us to Petra. The taxi ride took another 3 hours before we arrived at the historic city. The most famous attraction is, surprisingly, the treasury of the city, called Al-Khazneh.

Day 10-11: Flew to New Delhi and got a driver to take us to Agra, a four hour drive. The next day we visited the Taj Mahal, which is an Islamic mausoleum housing the tomb of the favorite wife of the Mughal emperor Shah Jahan. The construction was completed in 1643.

Day 12-13: Flew to Beijing and took a taxi to the Great Wall, the last wonder of our trip. The Great Wall was historically built to protect China against nomadic groups from the Eurasian Steppe.

a

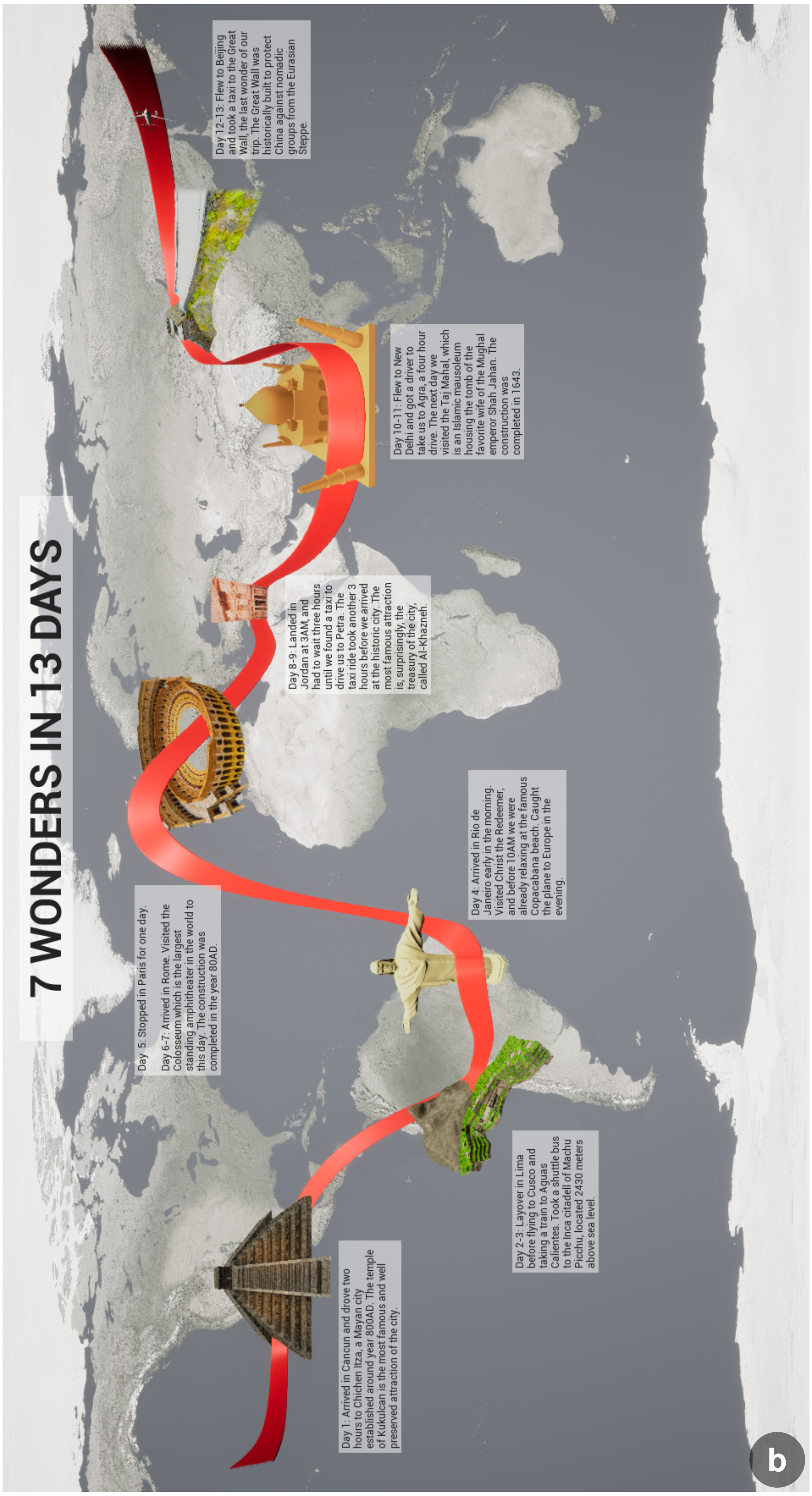

7 WONDERS IN 13 DAYS
Day 1: Arrived in Cancun and drove two hours to Chichen Itza, a Mayan city established around year 800AD. The temple of Kukulcan is the most famous and well preserved attraction of the city.
Day 2-3: Layover in Lima before flying to Cusco and taking a train to Aguas Calientes. Took a shuttle bus to the Inca citadell of Machu Picchu, located 2430 meters above sea level.
Day 4: Arrived in Rio de Janeiro early in the morning. Visited Christ the Redeemer, and before 10AM we were already relaxing at the famous Copacabana beach. Caught the plane to Europe in the evening.
Day 5: Stopped in Paris for one day.
Day 6-7: Arrived in Rome. Visited the Colosseum which is the largest standing amphitheater in the world to this day. The construction was completed in the year 80AD.
Day 8-9: Landed in Jordan at 3AM, and had to wait three hours until we found a taxi to drive us to Petra. The taxi ride took another 3 hours before we arrived at the historic city. The most famous attraction is, surprisingly, the treasury of the city, called Al-Khazneh.
Day 10-11: Flew to New Delhi and got a driver to take us to Agra, a four hour drive. The next day we visited the Taj Mahal, which is an Islamic mausoleum housing the tomb of the favorite wife of the Mughal emperor Shah Jahan. The construction was completed in 1643.
Day 12-13: Flew to Beijing and took a taxi to the Great Wall, the last wonder of our trip. The Great Wall was historically built to protect China against nomadic groups from the Eurasian Steppe.
b

Figure 5.2: Two snapshots of the poster titled *7 Wonders in 13 Days*. In the top image (a), the animated airplane following the focus object is close to the Colosseum in the fourth scene, whereas in the bottom image (b), the airplane has passed the Great Wall of China in the final scene.

Chapter 6

Discussion and Limitations

Based on the expert interview in Section 5.3, we can see that the linear structure of our pipeline might be too strict. The point that was made about how the global layout and format, i.e., length of width and height, can influence how the local layout is set up was particularly interesting. With this perspective, we think that the framework could be adapted such that between the local and global layout steps, there is a circular pipeline, whereas the detailed layout should remain in the linear pipeline as this step is always performed last. Another option could be to add a sketching or drafting step before the local layout step, where the author can conceptualize the poster to guide the local layout. The expert also pointed out that although the narrative flow is clear, the direction of this flow is not necessarily clear without an additional visual cue, such as the animated airplane in our second usage example. While the expert interview identified some areas that could be improved, it also verified that the pipeline was meaningful, although possibly too strict, and that the posters created with our framework have visual interest and a clear narrative structure.

Although using UE came with many benefits with regards to rendering power and a large API, it also added some limitations for the implementation that were undesirable. One of these limitations is related to the textual elements that the author can add during the detailed layout step of the pipeline. In UE, a font used in the GUI cannot be changed during runtime. This means that the fonts must be determined, as well as their size, color, justification, and emphasis, before the author enters the runtime mode, i.e., the global and detailed layout steps. Because of this we added only two types of textual elements: annotations with a small font size, black color, left justification and no emphasis, and titles with a large font size, black color, central justification and bold emphasis. Since the color of the font cannot be changed during runtime, we implemented the option to add a semi-translucent white background to the text box so that there is sufficient contrast to the black text on a darker section of the poster. The effect of the textual elements could be enriched by being able to, for example, make some words in the annotations bold or italic to emphasize them, or adapt the color of the text to enhance the contrast to the background.

Another limitation created by UE is related to saving the poster, specifically saving the finished poster as an image or video. Since posters are normally presented in a large format, e.g., printed on paper size A0 which corresponds to 84.1cm x 118.9cm, it is important that the image our tool produces is of high resolution. UE provides a possibility to save high resolution screen shots in runtime mode, but since these screen shots are rendered earlier than GUI elements in the rendering pipeline, they do not contain the textual elements added in the detailed layout step. To include the textual elements, we must use the regular screen shot of UE, which uses only the screen resolution. The resulting image, therefore, depends on the resolution of the author's screen. Figure 6.1 shows a comparison between a regular screen shot and a high resolution screen shot, both taken of the same poster on a screen with resolution 1920x1080 pixels.

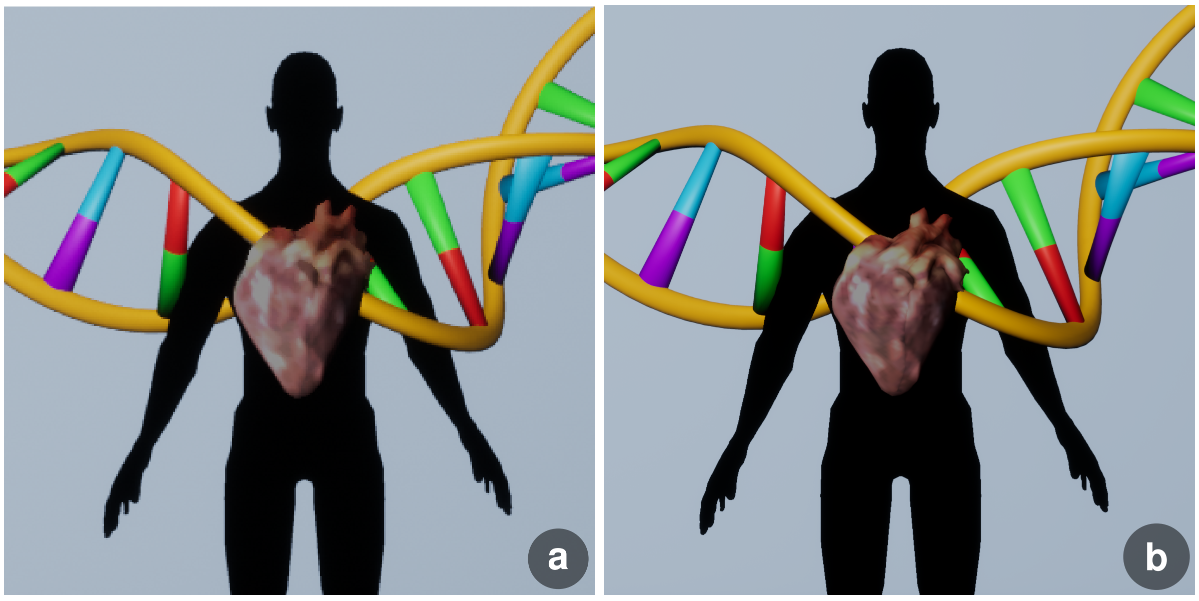


Figure 6.1: Comparison of (a) a regular screen shot and (b) a high resolution screen shot with a multiplier of 7, i.e., the resolution of the screen shot is 7 times larger than the resolution of the screen, which was the highest multiplier possible on our hardware.

In Figure 6.1 a the edges of the DNA and human models are jagged and they appear blurred compared to Figure 6.1 b. This blurring can also clearly be seen when comparing the texture of the heart model in the two images.

A limitation that was introduced by design is that the author can choose only one mesh for the focus object that is repeated for each spline segment. This means that the focus object mesh has to be repeatable, i.e., the end of the mesh must match the beginning of the mesh. It also means that the focus object is uniform throughout the poster which removes some design possibilities. The spline object and mesh in UE do not have this limitation, a different mesh can be set for each spline segment. It was introduced because it has certain benefits, for example, it simplifies the necessary interactions that the author must perform to define the mesh of the spline, and lets the author reuse one mesh instead of having to create several meshes that match each other perfectly. Despite this, there are cases where being able to define different meshes for the spline segments is more beneficial. For example, if the author wishes to create a poster of an elongated object where the scenes should explain different sections of this object. A concrete use case could be to create a poster about the worm, *Caenorhabditis elegans*, which is the subject of the OpenWorm project [34]. Here the mesh of the focus object would then be the whole worm, and with the current implementation, the poster would contain one worm per spline segment. By being able to select a different mesh per spline segment, one could section the worm mesh and use one section per spline segment. The focus object would then be one continuous worm instead of a sequence of connected worms.

Chapter 7

Conclusion and Future Work

We have created TimeBender, a framework and a prototype application for authoring 3D space-time posters. This framework is built on a 3-step pipeline which includes the local layout, global layout, and detailed layout steps. The prototype of the authoring framework is interactive, i.e., it runs with interactive frame rates for the posters we created. We evaluated our approach, and two example results produced using our prototype application, by an expert interview with a trained illustrator. Based on this evaluation we can conclude that the steps in our pipeline are meaningful, although a strictly linear pipeline, notably with regards to the first two steps: local and global layout, does not capture the regular workflow of an illustrator. The example posters we produced had features and techniques used by traditional illustrators, including fading effects, organic shapes, coloring to emphasize salient features, and animation to add interest and guide focus. This means that our framework is capable of producing posters that tell a story with data while using techniques inspired by traditional illustration to create communicative visualizations.

Although our viewpoint based approach to modelling the focus object did not produce the desired results, as described in Section 3.2.1, we think it should not be excluded in future work. Potentially, there is a different way of defining matched points which would produce a larger amount, i.e., more than one point per scene. With a larger set of matched points to use as input one could combine the pose recovery algorithm with the RANSAC algorithm [33] to handle the noise and yield more stable results.

Posters created with our framework do not support interaction from the audience. Research has indicated that being able to interact with a narrative visualization can lead to more trust in the presented results as the audience can verify the claims themselves in the data [77]. Since the scenes within a poster are rendered dynamically, e.g., to allow for animated models, it would be possible to extend the framework with an interaction where the audience of the poster could move the scene cameras around to change the view of the scene in the poster. Other interactions could be to, for example, select a scene of interest to get more details about this scene or activate animation of the models within this scene.

Another future extension of the framework could be to automate parts of the pipeline, e.g., the global layouting. The author could, for example, sketch a path they would like the focus object to follow and the system would place the scenes along this path. Another solution could be to create layouts in a force-based way, similar to force-based graph drawing, where the author could specify the importance of each scene which would correlate to the strength of a force pulling them towards the camera, i.e., the higher the importance of a scene, the stronger this force and thus the larger the scene would be in the poster. Attraction forces could be placed between the neighbouring scenes in the narrative, and distance-dependent repulsion forces between all scenes to ensure none of them overlap.

List of Acronyms and Abbreviations

API	Application Programming Interface.
FPS	Frames per Second.
GUI	Graphical User Interface.
SCC	SceneCaptureComponent2D.
UE	Unreal Engine.

Bibliography

[1] Rikke Aas. 7 Wonders in 13 Days - Animated. https://youtu.be/G9SpR5T14FY, .

[2] Rikke Aas. The Genetic Journey - Animated. https://youtu.be/DHiMsOAm_2o, .

[3] Adobe. After Effects. https://www.adobe.com/products/aftereffects.html, .

[4] Adobe. Photoshop. https://www.adobe.com/no/products/photoshop.html, .

[5] ajmalsaleem88. Colosseum_Rome_Italy. https://skfb.ly/ot6XK. This work is licensed under the Creative Commons Attribution 4.0 International License. To view a copy of this license, visit http://creativecommons.org/licenses/by/4.0/.

[6] Scott Bateman, Regan L. Mandryk, Carl Gutwin, Aaron Genest, David McDine, and Christopher Brooks. Useful Junk? The Effects of Visual Embellishment on Comprehension and Memorability of Charts. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, page 2573–2582, 2010. doi: 10.1145/1753326.1753716.

[7] Tobias Ritschel Bernhard Reinert and Hans-Peter Seidel. Homunculus Warping: Conveying Importance using Self-intersection-free Non-homogeneous Mesh Deformation. *Computer Graphics Forum*, 7(31):2165–2171, 2012. doi: 10.1111/j.1467-8659.2012.03209.x.

[8] Blender. Blender. https://www.blender.org/.

[9] Michelle A. Borkin, Azalea A. Vo, Zoya Bylinskii, Phillip Isola, Shashank Sunkavalli, Aude Oliva, and Hanspeter Pfister. What Makes a Visualization Memorable? *IEEE Transactions on Visualization and Computer Graphics*, 19(12):2306–2315, 2013. doi: 10.1109/TVCG.2013.234.

[10] Adrien Bousseau, Matt Kaplan, Joëlle Thollot, and François X. Sillion. Interactive Watercolor Rendering with Temporal Coherence and Abstraction. In *Proceedings of*

the International Symposium on Non-Photorealistic Animation and Rendering, page 141–149, 2006. doi: 10.1145/1124728.1124751.

[11] Andrea Brambilla, Robert Carnecky, Ronald Peikert, Ivan Viola, and Helwig Hauser. Illustrative Flow Visualization: State of the Art, Trends and Challenges. In *Proceedings of Eurographics - State of the Art Reports*, pages 69–97, 2012. doi: 10.2312/conf/EG2012/stars/075-094.

[12] Stefan Bruckner and Eduard Gröller. VolumeShop: An Interactive System for Direct Volume Illustration. In *Proceedings of the IEEE Visualization Conference*, pages 671–678, 2005. doi: 10.1109/VISUAL.2005.1532856.

[13] Stefan Bruckner and Eduard Gröller. Exploded Views for Volume Data. *IEEE Transactions on Visualization and Computer Graphics*, 12(5):1077–1084, 2006. doi: 10.1109/TVCG.2006.140.

[14] Stefan Bruckner, Sören Grimm, Armin Kanitsar, and M. Eduard Gröller. Illustrative Context-Preserving Volume Rendering. In *Proceedings of the IEEE VGTC Symposium on Visualization*, pages 1559–1569, 2005. doi: 10.2312/VisSym/EuroVis05/069-076.

[15] Stefan Bruckner, Peter Rautek, Ivan Viola, Mike Roberts, Mario Costa Sousa, and M. Eduard Gröller. Hybrid visibility compositing and masking for illustrative rendering. *Computers & Graphics*, 34(4):361–369, 2010. doi: 10.1016/j.cag.2010.04.003.

[16] Chris Bryan, Kwan-Liu Ma, and Jonathan Woodring. Temporal Summary Images: An Approach to Narrative Visualization via Interactive Annotation Generation and Placement. *IEEE Transactions on Visualization and Computer Graphics*, 23(1): 511–520, 2017. doi: 10.1109/TVCG.2016.2598876.

[17] Robert Carnecky, Raphael Fuchs, Stephanie Mehl, Yun Jang, and Ronald Peikert. Smart Transparency for Illustrative Visualization of Complex Flow Surfaces. *IEEE Transactions on Visualization and Computer Graphics*, 19(5):838–851, 2013. doi: 10.1109/TVCG.2012.159.

[18] M. Sheelagh T. Carpendale, D.J. Cowperthwaite, and F.D. Fracchia. Distortion Viewing Techniques for 3-Dimensional Data. In *Proceedings IEEE Symposium on Information Visualization*, pages 46–53, 1996. doi: 10.1109/INFVIS.1996.559215.

[19] Cheng-Kai Chen, Shi Yan, Hongfeng Yu, Nelson Max, and Kwan-Liu Ma. An Illustrative Visualization Framework for 3D Vector Fields. *Computer Graphics Forum*, 30(7):1941–1951, 2011. doi: 10.1111/j.1467-8659.2011.02064.x.

[20] Siming Chen, Jie Li, Gennady Andrienko, Natalia Andrienko, Yun Wang, Phong H. Nguyen, and Cagatay Turkay. Supporting Story Synthesis: Bridging the Gap between Visual Analytics and Storytelling. *IEEE Transactions on Visualization and Computer Graphics*, 26(7):2499–2516, 2020. doi: 10.1109/TVCG.2018.2889054.

[21] Ming-Te Chi and Tong-Yee Lee. Stylized and Abstract Painterly Rendering System Using a Multiscale Segmented Sphere Hierarchy. *IEEE Transactions on Visualization and Computer Graphics*, 12(1):61–72, 2006. doi: 10.1109/TVCG.2006.14.

[22] choi464. Taj Mahal. `https://skfb.ly/6UPrL`. This work is licensed under the Creative Commons Attribution 4.0 International License. To view a copy of this license, visit `http://creativecommons.org/licenses/by/4.0/`.

[23] Wei-Ta Chu, Chia-Hsiang Yu, and Hsin-Han Wang. Optimized Comics-Based Storytelling for Temporal Image Sequences. *IEEE Transactions on Multimedia*, 17(2): 201–215, 2015. doi: 10.1109/TMM.2014.2383616.

[24] Carlos Correa, Deborah Silver, and Min Chen. Feature Aligned Volume Manipulation for Illustration and Visualization. *IEEE Transactions on Visualization and Computer Graphics*, 12(5):1069–1076, 2006. doi: 10.1109/TVCG.2006.144.

[25] Carlos D. Correa and Kwan-Liu Ma. Dynamic Video Narratives. *ACM Transactions on Graphics*, 29(3):1–9, 2010. doi: 10.1145/1778765.1778825.

[26] Balázs Csébfalvi, Lukas Mroz, Helwig Hauser, Andreas König, and Eduard Gröller. Fast Visualization of Object Contours by Non-Photorealistic Volume Rendering. *Computer Graphics Forum*, 20(3):452–460, 2001. doi: 10.1111/1467-8659.00538.

[27] DeepDreamDimension. 3D model BASE MESH—Man Simple. `https://www.turbosquid.com/3d-models/3d-model-base-mesh-simple-man-1550153`. This work is licensed under the 3D Model License:Standard by TurboSquid. To view a copy of this license, visit `https://blog.turbosquid.com/turbosquid-3d-model-license/`.

[28] Diego Delso. Petra. `https://commons.wikimedia.org/wiki/File:The_Treasury,_Petra,_Jordan5.jpg`. This work is licensed under the Creative Commons Attribution-ShareAlike 3.0 International License. To view a copy of this license, visit `https://creativecommons.org/licenses/by-sa/3.0`.

[29] Sara J ElShafie. Making Science Meaningful for Broad Audiences through Stories. *Integrative and Comparative Biology*, 58(6):1213–1223, 2018. doi: 10.1093/icb/icy103.

[30] Eric Enderton, Erik Sintorn, Peter Shirley, and David Luebke. Stochastic Transparency. *IEEE Transactions on Visualization and Computer Graphics*, 17(8):1036–1047, 2011. doi: 10.1109/TVCG.2010.123.

[31] Kaveh Fathian and Nicholas R. Gans. A New Approach for Solving the Five-Point Relative Pose Problem for Vision-Based Estimation and Control. In *Proceedings of the American Control Conference*, pages 103–109, 2014. doi: 10.1109/ACC.2014.6859364.

[32] Stephen Few. The Chartjunk Debate: A Close Examination of Recent Findings. `http://www.perceptualedge.com/articles/visual_business_intelligence/the_chartjunk_debate.pdf`, 2011. Accessed: 22-06-2022.

[33] Martin A. Fischler and Robert C. Bolles. Random Sample Consensus: A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography. *Communications of the ACM*, 24(6):381–395, 1981. doi: 10.1145/358669.358692.

[34] OpenWorm Foundation. OpenWorm. `https://https://openworm.org/`. Accessed: 15-08-2022.

[35] Epic Games. Unreal Engine. `https://www.unrealengine.com`.

[36] Laura Garrison, Monique Meuschke, Jennifer Fairman, Noeska N. Smit, Bernhard Preim, and Stefan Bruckner. An Exploration of Practice and Preferences for the Visual Communication of Biomedical Processes. In *Proceedings of the Eurographics Workshop on Visual Computing for Biology and Medicine*, pages 1–12, 2021. doi: 10.2312/vcbm.20211339.

[37] Laura A. Garrison, Ivan Kolesar, Ivan Viola, Helwig Hauser, and Stefan Bruckner. Trends & Opportunities in Visualization for Physiology: A Multiscale Overview. *Computer Graphics Forum*, 41(3):609–643, 2022. doi: 10.1111/cgf.14575.

[38] Gerald Geake. Interface lg. `https://commons.wikimedia.org/wiki/File:Interface_lg.jpg`. This work is licensed under the Creative Commons Attribution-ShareAlike 3.0 International License. To view a copy of this license, visit `http://creativecommons.org/licenses/by-sa/3.0/`.

[39] Amy Gooch, Bruce Gooch, Peter Shirley, and Elaine Cohen. A Non-Photorealistic Lighting Model for Automatic Technical Illustration. In *Proceedings of the Conference on Computer Graphics and Interactive Techniques*, page 447–452, 1998. doi: 10.1145/280814.280950.

[40] Sarkis Halladjian, Haichao Miao, David Kouřil, M. Eduard Gröller, Ivan Viola, and Tobias Isenberg. Scale Trotter: Illustrative Visual Travels Across Negative Scales. *IEEE Transactions on Visualization and Computer Graphics*, 26(1):654–664, 2020. doi: 10.1109/TVCG.2019.2934334.

[41] Richard I. Hartley. In Defense of the Eight-Point algorithm. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19(6):580–593, 1997. doi: 10.1109/34.601246.

[42] Wei-Hsien Hsu, Kwan-Liu Ma, and Carlos Correa. A Rendering Framework for Multiscale Views of 3D Models. *ACM Transactions on Graphics*, 30(6):1–10, 2011. doi: 10.1145/2070781.2024165.

[43] Jessica Hullman and Nick Diakopoulos. Visualization Rhetoric: Framing Effects in Narrative Visualization. *IEEE Transactions on Visualization and Computer Graphics*, 17(12):2231–2240, 2011. doi: 10.1109/TVCG.2011.255.

[44] jhonpersonvl. Animal Cell. `https://free3d.com/3d-model/clula-animal-758429.html`. This work is licensed under the Personal Use License of Free3D. To view a copy of this license, visit `https://free3d.com/royalty-free-license`.

[45] T. A. Keahey and E. L. Robertson. Techniques for Non-Linear Magnification Transformations. In *Proceedings of the IEEE Symposium on Information Visualization*, pages 38–45, 1996. doi: 10.5555/857187.857606.

[46] Nam Wook Kim, Eston Schweickart, Zhicheng Liu, Mira Dontcheva, Wilmot Li, Jovan Popovic, and Hanspeter Pfister. Data-Driven Guides: Supporting Expressive Design for Information Graphics. *IEEE Transactions on Visualization and Computer Graphics*, 23(1):491–500, 2017. doi: 10.1109/TVCG.2016.2598620.

[47] Gordon Kindlmann, Ross Whitaker, Tolga Tasdizen, and T. Moller. Curvature-Based Transfer Functions for Direct Volume Rendering: Methods and Applications. In *Proceedings of the IEEE Visualization Conference*, pages 513–520, 2003. doi: 10.1109/VISUAL.2003.1250414.

[48] R. Kosara, S. Miksch, and H. Hauser. Semantic Depth of Field. In *Proceedings of the IEEE Symposium on Information Visualization*, pages 97–104, 2001. doi: 10.1109/INFVIS.2001.963286.

[49] Robert Kosara. Presentation-Oriented Visualization Techniques. *IEEE Computer Graphics and Applications*, 36(1):80–85, 2016. doi: 10.1109/MCG.2016.2.

[50] Martin Kraus. Perspective view frustum. `https://upload.wikimedia.org/wikipedia/commons/9/90/Perspective_view_frustum.png`. This work is licensed under the Creative Commons Attribution-ShareAlike 3.0 International License. To view a copy of this license, visit `http://creativecommons.org/licenses/by-sa/3.0/`.

[51] Jens Krüger, Jens Schneider, and Rüdiger Westermann. ClearView: An Interactive Context Preserving Hotspot Visualization Technique. *IEEE Transactions on Visualization and Computer Graphics*, 12(5):941–948, 2006. doi: 10.1109/TVCG.2006.124.

[52] Kai Lawonn, Ivan Viola, Bernhard Preim, and Tobias Isenberg. A Survey of Surface-Based Illustrative Rendering for Visualization. *Computer Graphics Forum*, 37(6): 205–234, 2018. doi: 10.1111/cgf.13322.

[53] Lionel. Great Wall China (model 1). `https://skfb.ly/6q98F`. This work is licensed under the Creative Commons Attribution-NonCommercial 4.0 International License. To view a copy of this license, visit `http://creativecommons.org/licenses/by-nc/4.0/`.

[54] Aidong Lu, C.J. Morris, D.S. Ebert, P. Rheingans, and C. Hansen. Non-Photorealistic Volume Rendering Using Stippling Techniques. In *Proceedings of the IEEE Visualization Conference*, pages 211–218, 2002. doi: 10.1109/VISUAL.2002.1183777.

[55] Kwan-Liu Ma, Isaac Liao, Jennifer Frazier, Helwig Hauser, and Helen-Nicole Kostis. Scientific Storytelling Using Visualization. *IEEE Computer Graphics and Applications*, 32(1):12–19, 2012. doi: 10.1109/MCG.2012.24.

[56] Monique Meuschke, Laura A. Garrison, Noeska N. Smit, Benjamin Bach, Sarah Mittenentzwei, Veronika Weiß, Stefan Bruckner, Kai Lawonn, and Bernhard Preim. Narrative medical visualization to communicate disease data. *Computers & Graphics*, 107:144–157, 2022. doi: 10.1016/j.cag.2022.07.017.

[57] Mirfen. Pyramid. `https://skfb.ly/6DpT7`. This work is licensed under the Creative Commons Attribution 4.0 International License. To view a copy of this license, visit `http://creativecommons.org/licenses/by/4.0/`.

[58] Tamara Munzner. *Visualization Analysis and Design.* CRC Press, 2015.

[59] David Nister. An Efficient Solution to the Five-Point Relative Pose Problem. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 26(6):756–770, 2004. doi: 10.1109/TPAMI.2004.17.

[60] Linda Nye and the Exploratorium Visualization Laboratory. Zoom Into the Human Bloodstream. https://www.nsf.gov/news/mmg/mmg_disp.jsp?med_id=64550, 2009. This work is licensed under the Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License. To view a copy of this license, visit http://creativecommons.org/licenses/by-nc-sa/4.0/.

[61] Ji Hwan Park, Arie Kaufman, and Klaus Mueller. Graphoto: Aesthetically Pleasing Charts for Casual Information Visualization. *IEEE Computer Graphics and Applications*, 38(6):67–82, 2018. doi: 10.1109/MCG.2018.2879066.

[62] Paulmcg1. DNA Strand. https://www.turbosquid.com/3d-models/3d-dna-strand-model/247520. This work is licensed under the 3D Model License:Standard by TurboSquid. To view a copy of this license, visit https://blog.turbosquid.com/turbosquid-3d-model-license/.

[63] Weichao Qiu and Alan Yuille. UnrealCV: Connecting Computer Vision to Unreal Engine. In *Proceedings of the European Conference on Computer Vision Workshops*, pages 909–916, 2016. doi: 10.1007/978-3-319-49409-8_75.

[64] quadmade studio. Machu Picchu Wonder. https://www.turbosquid.com/3d-models/picchu-wonder-3ds/657133. This work is licensed under the 3D Model License:Standard by TurboSquid. To view a copy of this license, visit https://blog.turbosquid.com/turbosquid-3d-model-license/.

[65] Peter Rautek, Stefan Bruckner, Eduard Gröller, and Ivan Viola. Illustrative Visualization: New Technology or Useless Tautology? *ACM SIGGRAPH Computer Graphics*, 42(3):1–8, 2008. doi: 10.1145/1408626.1408633.

[66] Ringo3D. Earth. https://www.turbosquid.com/3d-models/earth-max-free/1016431, . This work is licensed under the 3D Model License:Standard by TurboSquid. To view a copy of this license, visit https://blog.turbosquid.com/turbosquid-3d-model-license/.

[67] Ringo3D. Heart. https://www.turbosquid.com/3d-models/free-heart-3d-model/957963, . This work is licensed under the 3D Model License:Standard by TurboSquid. To view a copy of this license, visit https://blog.turbosquid.com/turbosquid-3d-model-license/.

[68] Edward Segel and Jeffrey Heer. Narrative Visualization: Telling Stories with Data. *IEEE Transactions on Visualization and Computer Graphics*, 16(6):1139–1148, 2010. doi: 10.1109/TVCG.2010.179.

[69] Fairman Studios. The Genetic Journey. https://www.fairmanstudios.com/project/genetic-2/, 2019. Accessed: 01-09-2022.

[70] Megan Sullivan. How to Travel to the 7 Wonders of the World in 13 Days. https://medium.com/@megthelegend/how-to-travel-to-the-7-wonders-of-the-world-in-13-days-a-dirtbags-guide-a087da2f0089. Accessed: 12-08-2022.

[71] Edward R. Tufte. *The Visual Display of Quantitative Information.* Graphics Press, 2001.

[72] Claude Valette. Panneau Des Chevaux. https://upload.wikimedia.org/wikipedia/commons/9/92/13_PanneauDesChevaux%28D%C3%A9tail%29.jpg. This work is licensed under the Creative Commons Attribution-ShareAlike 4.0 International License. To view a copy of this license, visit http://creativecommons.org/licenses/by-sa/4.0/.

[73] Andrew Vande Moere and Helen Purchase. On the role of design in information visualization. *Information Visualization*, 10(4):356–371, 2011. doi: 10.1177/1473871611415996.

[74] Ivan Viola and Eduard Gröller. Smart Visibility in Visualization. In *Proceedings of the Workshop on Computational Aesthetics in Graphics, Visualization and Imaging*, pages 209–216, 2005. doi: 10.2312/COMPAESTH/COMPAESTH05/209-216.

[75] Lujin Wang, Joachim Giesen, Kevin McDonnell, Peter Zolliker, and Klaus Mueller. Color Design for Illustrative Visualization. *IEEE Transactions on Visualization and Computer Graphics*, 14(6):1739–46, 2008. doi: 10.1109/TVCG.2008.118.

[76] Yun Wang, Haidong Zhang, He Huang, Xi Chen, Qiufeng Yin, Zhitao Hou, Dongmei Zhang, Qiong Luo, and Huamin Qu. InfoNice: Easy Creation of Information Graphics. In *Proceedings of the CHI Conference on Human Factors in Computing Systems*, pages 1–12, 2018. doi: 10.1145/3173574.3173909.

[77] Michael Wohlfart and Helwig Hauser. Story Telling for Presentation in Volume Visualization. In *Proceedings of the IEEE VGTC Symposium on Visualization*, pages 91–98, 2007. doi: 10.2312/VisSym/EuroVis07/091-098.

[78] J. Yu and L. McMillan. A Framework for Multiperspective Rendering. In *Proceedings of the Eurographics Workshop on Rendering Techniques*, pages 61–68, 2004. doi: 10.2312/EGWR/EGSR04/061-068.

[79] Johannes Zander, Tobias Isenberg, Stefan Schlechtweg, and Thomas Strothotte. High Quality Hatching. *Computer Graphics Forum*, 23(3):421–430, 2004. doi: 10.1111/j.1467-8659.2004.00773.x.