Animating String Diagrams

Iain Weetman

4th Year Project Report
Cognitive Science
School of Informatics
University of Edinburgh

2024

Abstract

String diagrams are simple diagrams which mainly consist of basic shapes and lines but can be used as a formal graphical language to perform mathematical reasoning. They are often drawn using the TikZ text-based format. However, a series of string diagrams can be difficult to follow and would often be clearer as an animation. This project introduces a tool [Weetman, 2024] which uses the animation library Manim to convert a series of string diagrams written in TikZ to a video which animates changes between the diagrams. The tool is web-based to be convenient and remove the need for users to have any technical knowledge. Finally, a usability study was conducted to evaluate the tool. The results support the design decisions made throughout the project and offer many ideas for future work on the tool.

Research Ethics Approval

This project obtained approval from the Informatics Research Ethics committee.
Ethics application number: 837563
Date when approval was obtained: 2024-01-30
The participants' information sheet and a consent form are included in the appendix.

Declaration

I declare that this thesis was composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification except as specified.

(*Iain Weetman*)

Acknowledgements

I wish to express my gratitude to my supervisor, Chris Heunen, for his invaluable support throughout this project. His prompt email replies, insightful feedback, and assistance in contacting participants for the usability study have been immensely helpful. Moreover, his guidance on how to initiate and navigate the project from the outset has been instrumental in its completion.

I would also like to thank the participants of the usability study for giving up their time to provide helpful feedback on this project.

Table of Contents

Chapter 1

Introduction

String diagrams are graphical representations of equations, which are constructed from simple shapes and lines. A series of string diagrams can be difficult to follow and would often be clearer as an animation. However, during the research section of this project, no available tools were found to convert string diagrams in a useful format (e.g. TikZ) to an animation. Therefore, this project introduces a tool [Weetman, 2024] which creates an animation from a series of string diagrams. It may be useful for the reader to have a basic understanding of their purpose, but this is not essential as the majority of this project can be understood with no knowledge of string diagrams.

The main achievements of this project are the following:

- A tool was developed to create an animation from a series of string diagrams.
- A web-based front-end and back-end were developed and the tool was deployed online.
- A usability study was conducted and the results suggested that the main design decisions of the project maximise how useful the tool is for potential users.

Some challenges were encountered in the development of this project, such as converting lines and compass directions from TikZ to Manim, automatically matching lines between two consecutive diagrams, learning React and Django, and deploying the back-end of the system. Details of these challenges are discussed throughout the report.

1.1 String Diagrams

1.1.1 Equations vs Diagrams

Both diagrams and equations are well-suited to particular situations. Diagrams are used in many different subject areas to facilitate the communication of ideas (for example, diagrams can include Bayesian networks, neural networks, and electrical circuits). In some cases, diagrams may help a reader to understand a concept which is difficult to communicate through words. In contrast, equations are designed for mathematical reasoning, so can be conveniently combined and manipulated while seeking the solution

to a problem. However, equations may lack the clarity and ease of understanding gained through diagrams.

String diagrams combine the advantages of equations and diagrams. They are a graphical representation of equations which can be manipulated to find a solution to a problem. They represent complex equations with simple diagrams built from lines and nodes (i.e. basic shapes such as circles and rectangles). Manipulations of equations then becomes as simple as moving nodes and lines, while remaining just as rigorous.

1.1.2 String Diagram Uses

String diagrams were first introduced by Penrose [1971] for calculations with tensors, and Joyal and Street [1991] (as cited in Kissinger and Zamdzhiev [2015]) later showed that string diagrams could be used more generally, extending their possible uses from tensors to any group of processes with operations for parallel and sequential composition. While string diagrams are primarily used in category theory, they are also found in a wide variety of other fields, including finite-state automata [Piedeleu and Zanasi, 2023], functional programming [Riley, 2018], databases [Patterson, 2017], and natural language processing [Coecke et al., 2010].

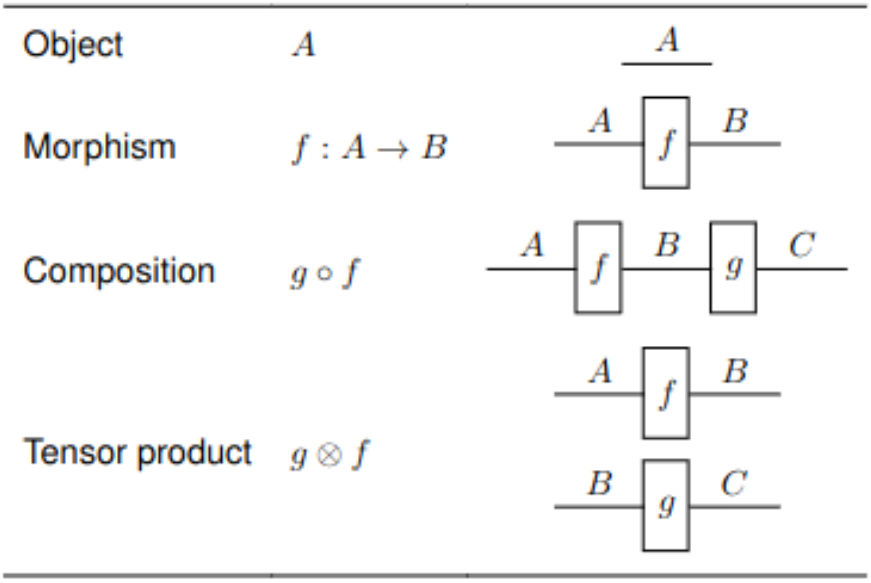


Figure 1.1: Category-theoretic notions represented using string diagrams [Selinger, 2010]

1.1.3 Reading String Diagrams

This section briefly describes how string diagrams are used to represent equations. String diagrams are mostly drawn using lines (also known as wires/edges) and nodes (also known as boxes), and are usually read from left to right. The category-theoretic notions known as objects, morphisms, composition, and the tensor product are represented in string diagrams as follows (also shown in figure 1.1):

- Objects are represented by lines.
- Morphisms are represented by nodes.
- Composition is represented by combining two morphisms sequentially, so that the output of one becomes the input of the other.
- The tensor product is represented by combining two morphisms in parallel.

1.1.4 Types of String Diagram

Since string diagrams are used in so many different areas, it is not surprising that there are many types of string diagram. These types can vary in restrictions on the diagrams (e.g. how wires can cross or the operations allowed) or can extend the basic diagrams described above with additional features. Selinger [2010] describes multiple extensions to these basic string diagrams for various aspects of category theory (shown graphically in figure 1.2). For example, braided monoidal categories include rules for the crossing of wires (braid and crossing in the figure), diagrams for autonomous categories allow wires to have 180-degree turns (unit and counit in the figure), diagrams for traced categories allow wires to loop if their endpoints retain a direction of left to right, product category diagrams include copy and erase/delete operations, and coproduct category diagrams include merge and initial/create operations. Despite the wide range of extensions, these are all formed from similar graphical components (lines, nodes, labels), so these types of string diagrams are the focus of the software tool introduced in this report. Selinger also discusses other extensions, such as braids and twists for balanced monoidal categories, but these are not considered in this project.

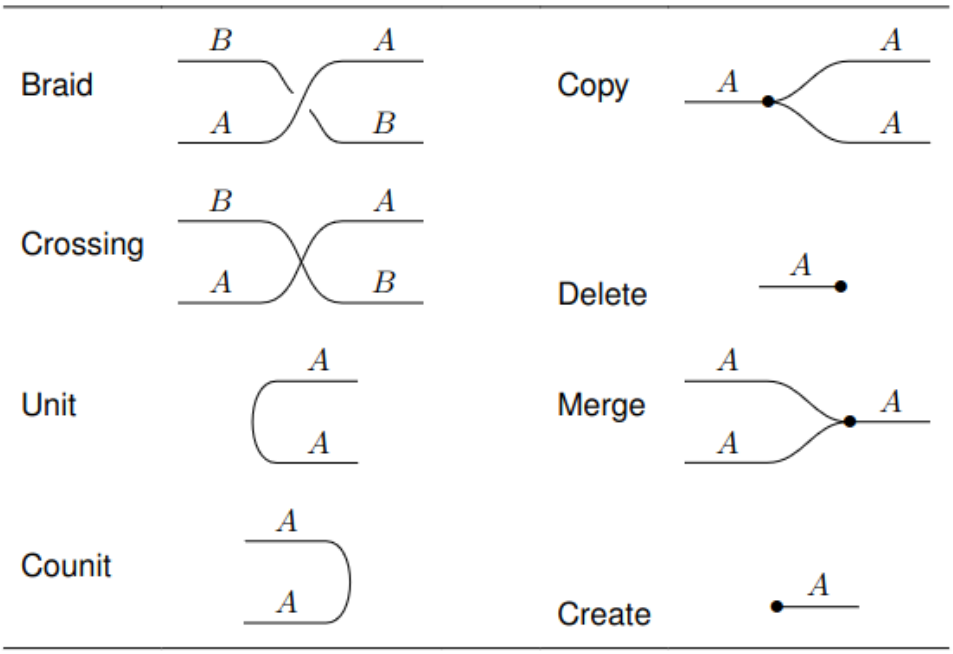


Figure 1.2: String diagram extensions [Madrigal, 2023]

1.2 Software tools for creating string diagrams

This section discusses the software tools which are already available for creating string diagrams. It considers TikZiT [Kissinger et al., 2020], quiver[varkor, 2020a], homotopy.io [nLab authors, 2023], Quantomatic [Kissinger and Zamdzhiev, 2015], FreeTikz [Heunen, 2018], a tool using the Haskell Diagrams library [Madrigal, 2023], Chyp [Kissinger, 2023], DisCoPy [de Felice et al., 2021], and TikzEdt [Willwacher, 2021]. This section will focus on how these tools are made available and how users create diagrams with these tools. Examining these aspects of available software tools provides valuable insights and helps to inform the design decisions of this project.

1.2.1 How tools are made available

The discussion below is summarised in table 1.1.

All of these tools are available to download as source code on GitHub. This is the easiest option for developers because code can simply be uploaded without the need for any consideration of user convenience. GitHub does not require a developer to include installation instructions, so there is no guarantee that users can easily install and use the tool without errors. In some cases, source code may be uploaded on GitHub simply because this provides a method for multiple developers to collaborate on a project.

TikZiT, Quantomatic, and TikzEdt are made available to install from their own websites. This often requires the consideration of multiple operating systems and may be difficult for developers to set up. Since these cases require more work from the developers to make the installation process simpler, users will likely find this method of installation easier than downloading the source code. However, this is not guaranteed and users could still encounter errors during installation, so may require support or technical knowledge.

Tool	Download source code	Install from website	Web-based	Programming language library
TikZiT	✓	✓		
quiver	✓		✓	
homotopy.io	✓		✓	
Quantomatic	✓	✓		
FreeTikZ	✓		✓	
Chyp	✓			✓
DisCoPy	✓			✓
Haskell Tool	✓			
TikzEdt	✓	✓		

Table 1.1: How tools are made available

FreeTikz, quiver, and homotopy.io are web-based. Web-based tools facilitate the creation of string diagrams by avoiding the need to install a tool on a user's computer. This also removes the risk of errors during installation. Bugs in the software are still possible, but these will no longer depend on the user's computer.

Chyp and DisCoPy differ from the other tools in that they are Python libraries. Once installed, Chyp uses a GUI in a similar way to the other tools without needing knowledge of Python. In contrast, DisCoPy is designed as a Python toolkit to be used within a Python program written by the user. These may be less convenient than a web-based tool, as users may still encounter errors during installation. However, since DisCoPy is the only tool designed to be used within a user's own program, it may be the most useful option for users who would like to integrate a tool into their own project.

1.2.2 Creating diagrams with these tools

This section discusses how users create diagrams with these tools. The methods used to create diagrams can be split into three categories: building diagrams using a GUI, using a programming language, and typing text in particular formats which represent diagrams. The methods discussed below are summarised in table 1.2.

1.2.2.1 Building diagrams with a GUI

The most common method used by these tools to create string diagrams is to provide an GUI editor for the user to build diagrams from components such as lines and circles. Taking this approach makes a tool accessible to more users, as no technical knowledge is required. This method is used by TikZiT, quiver, homotopy.io, and Quantomatic. Homotopy.io provides functionality for three-dimensional diagrams, but this is not necessary for the string diagrams we look at here. FreeTikz extends the use of a GUI editor to allow users to draw a string diagram with their mouse/stylus/finger and automatically convert this drawing to the required diagram in TikZ code, making the process of drawing diagrams especially quick and convenient. For these reasons and the advantages of web-based tools mentioned in section 1.2.1, FreeTikz may be the best choice for quickly creating string diagrams. Example FreeTikz input and output is shown in figure 1.3.

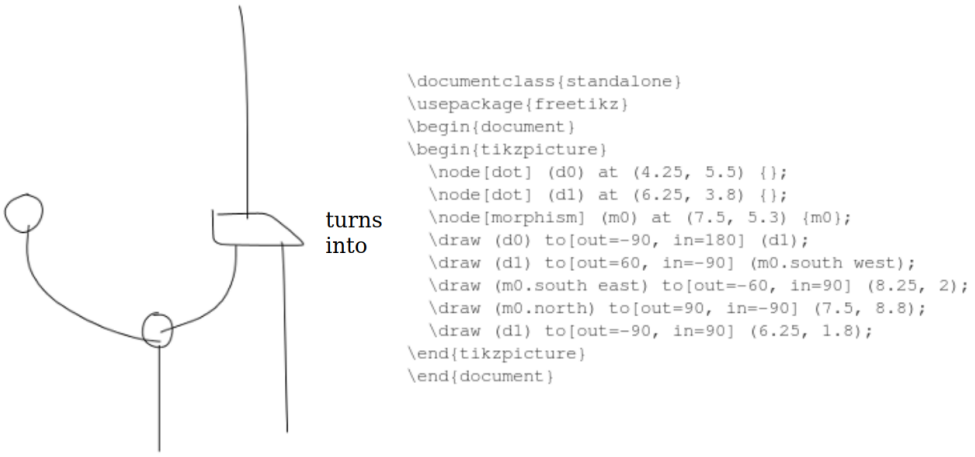


Figure 1.3: FreeTikz drawn input and resulting TikZ code [Heunen, 2018]

Tool	GUI Editor	Draw using mouse/stylus/finger	Coded in programming language	Text format
TikZiT	✓			✓
quiver	✓			✓
homotopy.io	✓			
Quantomatic	✓			✓
FreeTikz	✓	✓		✓
Chyp			✓	
DisCoPy			✓	
Haskell Tool				✓
TikzEdt				✓

Table 1.2: How users create diagrams with available tools

1.2.2.2 Programming diagrams

DisCoPy and Chyp require the diagrams to be coded in a particular programming language. DisCoPy uses Python, which many users may already be familiar with. Therefore, this use of a common programming language makes DisCoPy accessible to a wider range of users. Example DisCoPy Python code and the resulting diagram is shown in figure 1.4. Chyp uses its own programming language, as it needs to include rules and rewrite functions to fulfill its main purpose as a theorem prover. This also allows an input format which is similar to equations. Therefore, Chyp may be the best tool for users who wish to convert equations to string diagrams. Example Chyp code and the resulting diagrams are shown in figure 1.5.

```
from discopy.monoidal import Ty, Box, Layer, Diagram

x, y, z = Ty('x'), Ty('y'), Ty('z')
f, g, h = Box('f', x, y @ z), Box('g', y, z), Box('h', z, z)

assert f >> g @ h == Diagram(
    dom=x, cod=z @ z, inside=(
        Layer(Ty(), f, Ty()),
        Layer(Ty(), g, z),
        Layer(z,    h, Ty())))

(f >> g @ h).draw()
```

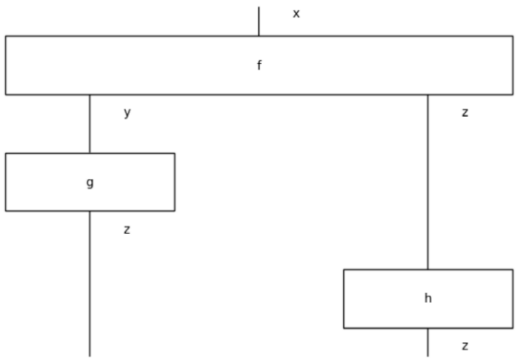
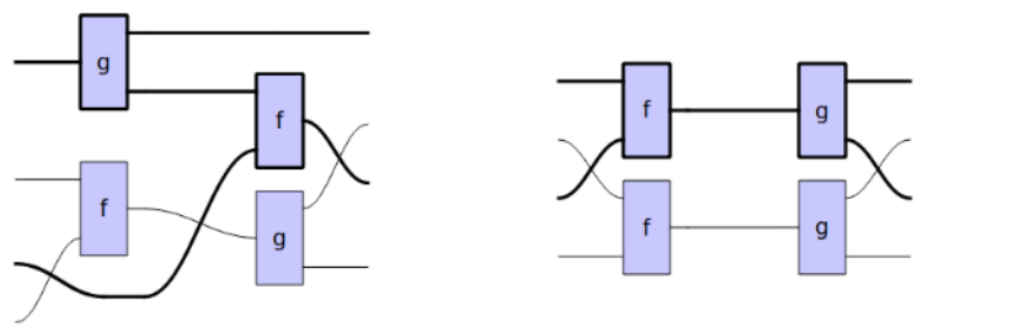


Figure 1.4: DisCoPy Python code and resulting diagram [de Felice et al., 2021]

1.2.2.3 Representing diagrams with text formats

Some tools allow users to input text in a particular format which describes the diagram. TikZiT allows users to input (and output) diagrams as TikZ code, although the TikZ code must be within the restricted subset of TikZ code which TikZiT uses. Example TikZiT TikZ code and the resulting diagram is shown in figure 1.6. Quiver allows users to import and export diagrams in `tikz-cd`, which is an extension of TikZ designed for commutative diagrams [Neves, 2011]. Quiver also allows users to export diagrams as an HTML `iframe` element which links to the diagram displayed on quiver. Example `tikz-cd` code and the resulting diagram in quiver is shown in figure 1.7.




```
let f2 = id * sw * id ; f * f
let g2 = g * g ; id * sw * id

rule bialg : f ; g = g * g ; id * sw *id ; f * f

rewrite ba2 :
  f * id ; f ; g ; g * id
  = f * g ; g * id * id ; id * sw * id ; f * f ; g * id by bialg

rewrite frob2:
   g2 * id * id ; id * id * f2
  = id * id * sw ; g * f * id ; id * id * sw ; id * f * g ; id * sw * id by frob
  = id * sw * id ; f * f ; g * g ; id * sw * id by frob
```

Figure 1.5: Chyp code and resulting diagrams [Kissinger, 2023]

Madrigal's Haskell tool takes JSON as input and renders this as a string diagram. However, it uses a specific set of attributes, so another application would need to use an identical set of JSON attributes for the two formats to be compatible. TikzEdt is a TikZ editor which provides users with real-time rendering of their TikZ code. However, TikzEdt may be unnecessary when using a tool such as TikZiT which can also render TikZ code (unless the TikZ accepted by TikZiT is insufficient).

```
\begin{tikzpicture}
   \begin{pgfonlayer}{nodelayer}
      \node [style=red dot] (0) at (-2, 0) {};
      \node [style=red dot] (1) at (2, 1) {};
      \node [style=red dot] (2) at (1, -1) {};
      \node [style=none] (3) at (-3, 0) {};
   \end{pgfonlayer}
   \begin{pgfonlayer}{edgelayer}
      \draw [in=-180, out=0, looseness=1.50] (0) to (1);
      \draw [bend right=60] (0) to (2);
      \draw (3.center) to (0);
   \end{pgfonlayer}
\end{tikzpicture}
```

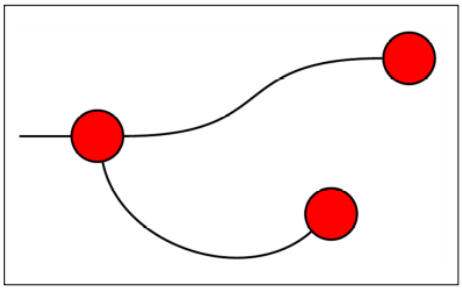


Figure 1.6: TikZiT TikZ code and resulting diagram

```
\begin{tikzcd}
   {(A \otimes I) \otimes B} && {A \otimes (I \otimes B)} \\
   & {A \otimes B}
   \arrow["{\rho_A \otimes B}"', from=1-1, to=2-2]
   \arrow["{A \otimes \lambda_B}", from=1-3, to=2-2]
   \arrow["{\alpha_{A, I, B}}", from=1-1, to=1-3]
\end{tikzcd}
```

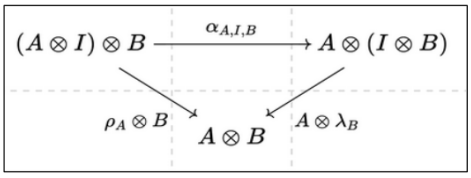


Figure 1.7: Quiver `tikz-cd` code and resulting diagram [varkor, 2020b]

Some tools use text-based descriptive formats but do not intend for users to input these formats in order to display a diagram. Homotopy.io allows users to import and export diagrams as JSON, but the JSON used contains only simple metadata and an identification code for the proof/diagram. Homotopy.io stores and retrieves a proof/diagram using this identification code, but other tools would not be able to import a diagram using this JSON as it does not contain information about how to draw the diagram. For this reason, it is not included in table 1.2. Quantomatic uses a complex set of folders for each project and diagrams are stored in QGraph format within one of these folders. The QGraph format is simply JSON with a specified set of attributes which Quantomatic uses. It uses a different set of attributes to Madrigal's Haskell tool, so JSON used by one tool would not be compatible with the other. It would be possible for other applications to use the same format, but this format is designed to be used within Quantomatic projects.

1.2.2.4 Connection to the software tool in this project

Each of the methods used to create diagrams have their own advantages:

- Building diagrams with a GUI editor is convenient and requires no technical knowledge.
- Using a programming language provides control and flexibility.
- Representing a diagram using a text format allows the user to import the diagram into other tools because text formats do not need to be specific to a single tool.

The application introduced in this report focuses on the animation of diagrams, so the application should allow users to animate diagrams which have been created in other tools. Building diagrams with a GUI editor and using a programming language are specific to the tool used. For example, the code used to program a diagram with DisCoPy is specific to DisCoPy. Only text formats of diagrams can be independent of the tool which created them, so this is the most suitable method to input diagrams into the application described in this report. Specific text formats which could be accepted are discussed in section 1.3.

1.3 Text Formats

There are many text formats which can be used to represent diagrams. For this project it was necessary to choose one specific format which could be used to input diagrams into the application. This section discusses the advantages and disadvantages of some of these formats (specifically TikZ, DOT, Asymptote, and JSON). The popularity of these diagram formats can be influenced by many factors, including the formats used in the tools discussed in the previous section. The discussion below is summarised in table 1.3.

TikZ [Tantau, 2007] is built on the PGF package and is used to create simple graphics inside LaTeX documents. It allows graphics to be created quickly with precise positioning and the use of macros. In addition, the main advantage in the case of this project is

that TikZ is widely used. TikZiT, FreeTikz, and DisCoPy all output TikZ, and quiver accepts `tikz-cd` which is an extension of TikZ. The disadvantages of TikZ include the difficulty to learn, the fact that compiling diagrams can be time-consuming, and that the code structure does not resemble the diagram structure [Tantau, 2007]. Some tools make the code structure more closely resemble the diagram structure, but these are often for specific types of diagram (for example, tree diagrams using Forest [Forest contributors, 2017]). TikZ is not the only LaTeX graphics package, but Tantau [2007] compares TikZ with other available LaTeX graphics packages and mentions the issues which TikZ solves, such as the lack of portability and inconsistent syntax of `pstricks`, the small size of `dratex`, and the difficulty including labels in `metapost`. For these reasons, this project considers TikZ rather than these other LaTeX graphics packages. The example TikZ code used in FreeTikz and TikZiT was shown in figures 1.3 and 1.6 respectively.

The second format to discuss is DOT, which was developed as part of Graphviz [Gansner et al., 2015] and is used by a few graphics tools. For example, `Canviz` [Canviz contributors] is a JavaScript library for rendering DOT graphs on a browser, `vis.js` [Vis.js contributors] is a JavaScript visualisation library which accepts the DOT format for particular graphics, and `d3-graphviz` [d3-graphviz contributors] converts DOT graphics to SVGs. One important advantage of the DOT format over TikZ is ease of use: for example, users will prefer to write $a \to b \to c \to a$ in DOT over creating three nodes and three edges separately in TikZ [Tantau, 2013]. There are also methods to convert the DOT format to TikZ, such as using `dot2tex` [Fauske, 2006] because TikZ allows mathematical notation, which is an advantage of TikZ over DOT. An example of a DOT diagram is shown in figure 1.8.

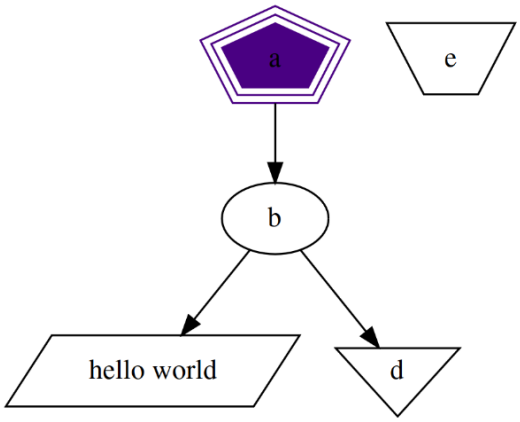


Figure 1.8: DOT diagram from the following source [Gansner et al., 2015]:

```
digraph G {
a -> b -> c;
b -> d;
a [shape=polygon,sides=5,peripheries=3,color=lightblue,style=filled];
c [shape=polygon,sides=4,skew=.4,label="hello world"]
d [shape=invtriangle];
e [shape=polygon,sides=4,distortion=.7];
}
```

```
picture intersection;
fill(intersection,c1,lightred+lightgreen);
clip(intersection,c2);
add(intersection);

draw(c1);
draw(c2);

label("$A$",z1);
label("$B$",z2);
path g=(0,-2)--(0,-0.25);
draw(Label("$A\cap B$",0),g,Arrow);
```

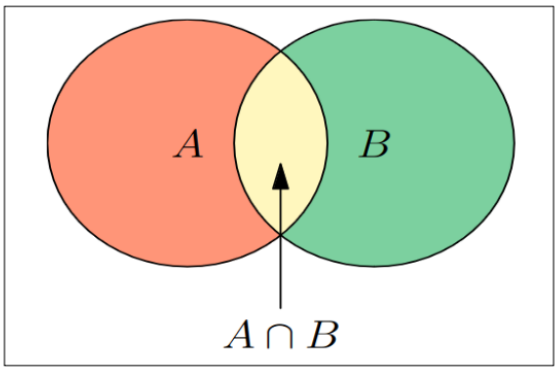


Figure 1.9: Asymptote source and diagram [Bowman and Hammerlindl, 2008]

The third format we will discuss here is the Asymptote vector graphics language [Bowman and Hammerlindl, 2008]. Asymptote has a similar syntax to C++ or Java, which facilitates the learning process and allows the user to write more complex program logic [Bowman, 2010]. Asymptote also automatically sizes pictures, allows three-dimensional graphics, and offers some animation functionality. Similar to TikZ, there is a LaTeX package which allows Asymptote code to be written inside a LaTeX document. An example of an Asymptote diagram is shown in figure 1.9.

Unlike the first three formats, JSON is not used exclusively for graphics purposes, but rather as a general format to store any data. Jackson [2016] discusses some of the advantages of JSON. These include the fact that it is not specific to a single programming language, it is simple and lightweight, and it can contain arrays of objects. JSON can be processed faster than other formats used for similar purposes such as XML [Yusof and Man, 2017], which may be important for applications which deal with large quantities of data. JSON is widely used, particularly easy to read and understand, and has a familiar JavaScript-based syntax (Bassett [2015] as cited in Houwink [2022]). These advantages could be the reasons for the choice of JSON to store diagrams in Quantomatic and Madrigal's Haskell tool. However, one major disadvantage of JSON is the variability between applications. The meaning of the other three formats does not vary between applications, but the set of JSON attributes used to represent diagrams is not standardised, so two JSON diagrams from different applications will likely be incompatible. For example, Quantomatic and Madrigal's Haskell tool do not use the same JSON attributes to store diagrams, so a diagram created with one tool could not be directly used with the other.

For this project, TikZ was chosen as the most suitable text format. The main reason for this decision was the popularity of TikZ because selecting a popular format increases the chance that this project will be useful to potential users. JSON is also popular, but the high probability that JSON from different tools is incompatible makes JSON representations of diagrams specific to the application which creates them.

Format	Widely used	Can be written inside LaTeX	Code structure represents diagram structure	Easy to read	Familiar syntax
TikZ	✓	✓			
DOT			✓	✓	
Asymptote		✓			✓
JSON	✓			✓	✓

Table 1.3: String diagram formats

1.4 Animation

The software tool introduced in this project animates string diagrams. In some cases, an animation may help learner understanding and be more effective at conveying an idea than a static diagram [Mayer, 2002, Berney and Bétrancourt, 2016]. The benefits of animation over static graphics may be even more noticeable when replacing a series of string diagrams. Despite the fact that string diagrams may be easier to understand than equations, a series of string diagrams could be difficult to follow. This section discusses some of the available animation packages which could be used to replace a series of diagrams with an animation. These are divided into two types: those which can be installed and run locally and those which are embedded into web pages. The discussion below is summarised in table 1.4.

1.4.1 Tools for creating animations locally

The first tool to consider is Manim [Manim contributors, 2023], which was created by Grant Sanderson for the YouTube channel 3Blue1Brown. Manim is a package for Python which is a popular language many users may already be familiar with. Since it was originally built for creating mathematical animations, the package has a mathematical focus and easily integrates LaTeX into any animation. However, there are also a few disadvantages to Manim. There are several dependencies which need to be installed before Manim can be used, which increases the chance of errors and can make deployment of an application significantly more complex (as shown in section 3.1.5.3). Manim's rendering is slow [Liu and Sun, 2023], but it can show a preview of the final frame without rendering the whole animation and offers configuration options to change the quality of the animation as low quality animations are faster to render. Because of Manim's popularity, multiple other tools have been developed to take advantage of Manim's animation capabilities [Eertmans, 2023][Huang et al., 2023][Yang et al., 2022].

The AAnim tool [Liu and Sun, 2023] is also based on Manim and makes improvements for use in computer science. It addresses the difficulty of learning to use Manim by simplifying user interaction and allowing users to create animations without a need for extensive knowledge in animation. It also adds functionality tailored to computer science concepts, such as automatically laying out animations for common data structures. AAnim focuses on algorithms and data structures, making the creation of these animations easy, but does not help with other types of animation, such as string diagrams.

Format	Produces video file	Used without many dependencies	Easy to integrate into project	Flexible	Completely free
Manim	✓		✓	✓	✓
DsDraw	✓	✓		✓	✓
CSS APIs		✓	✓		✓
SVGs with SMIL		✓	✓		✓
SVGs with JavaScript		✓	✓	✓	✓
HTML with JavaScript		✓	✓	✓	✓
GSAP	✓	✓	✓	✓	
D3.js		✓	✓	✓	✓

Table 1.4: Animation tools

DsDraw [Jeffries et al., 2020] is a lightweight web application which allows users to draw and animate graphics, using both a GUI and its own C-like programming language. In addition, it provides functionality to add audio to animations and combine animation recordings. DsDraw was designed to help with teaching computer science theory and contains many built-in types of diagram, which could be particularly convenient for users who need computer science diagrams. It would be possible to build and animate string diagrams on dsDraw without needing any additional software, but it would not be possible to input previously-built diagrams in any of the formats discussed in section 1.3. Since dsDraw is a full web-based animation application, it would also be difficult to integrate into a larger project in order to allow users to input these formats. Despite being web-based, it is not designed for helping users to embed animations in web pages, so was included in this section.

1.4.2 Tools for embedding animation in web pages

Many tools have been developed to embed animations in web pages. While some may only be useful for animating particular elements on web pages, others can be used to produce high-quality animations similar to those produced by the tools discussed above.

Garaizar et al. [2014] discusses many of the options for animation on websites. CSS provides a Transitions API and a Transforms API which can be used to create animations. An advantage of this method is that it provides information about the animation to the browser so the browser can control how the animation is run, but it also has a lack of flexibility. Animating SVGs with SMIL has the same advantage and disadvantage. Animating SVGs with JavaScript overcomes this issue and works at a high resolution with low resources, but JavaScript timers are often not accurate enough and the animation can become slow when an SVG contains many shapes. The HTML Canvas element with JavaScript allows users to create more complex, mathematical animations, but has the same disadvantages as animating SVGs with JavaScript.

JavaScript animation libraries which are more complex are also available. GSAP [GreenSock, 2023] can create SVG animations and interactive animations. It also supports storing, pausing, and modifying transitions within animations [Göring, 2023]. However, there are restrictions on the free version of GSAP. D3.js [Bostock and Observable, Inc., 2023] is free library designed for many types of data visualisation, but, unlike GSAP, it does not support storing, pausing, or modifying transitions.

Chapter 2

Design

This chapter discusses the design of the three main sections of the project: parsing, converting values between display formats, and animation. The system was split into these three sections for the purpose of modularity and to allow for the extension of the system through new parsers or animation methods. Figure 2.1 shows a diagram of the system architecture.

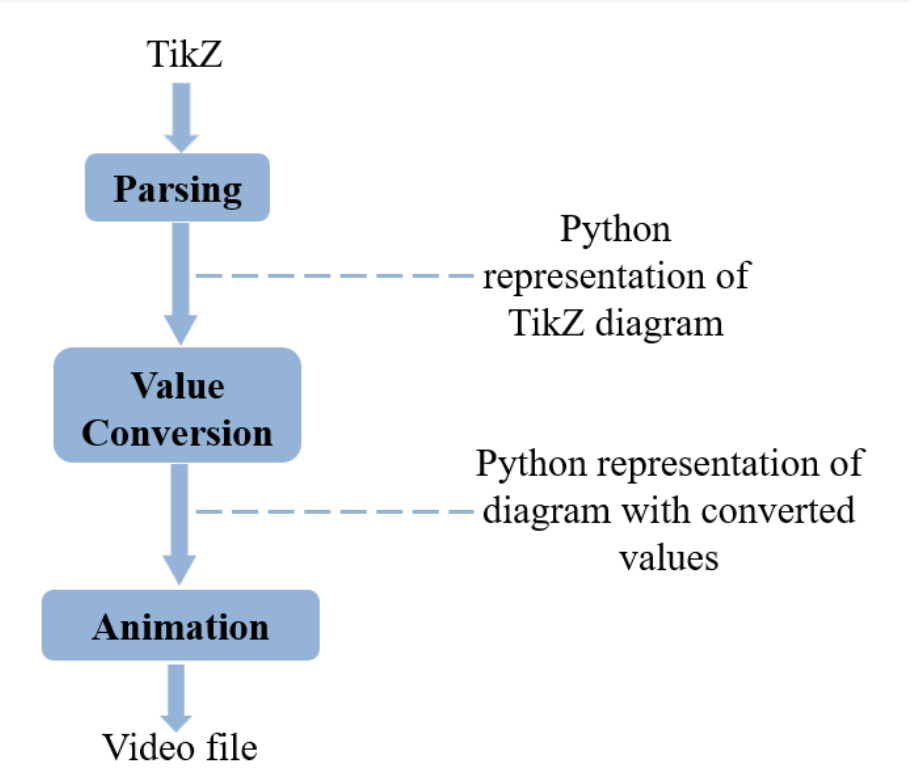


Figure 2.1: Diagram of application architecture

2.1 Parsing

2.1.1 Design decisions

The first decision to make in relation to parsing is the input format to be accepted by the system. The advantages and disadvantages of four possible formats (TikZ, DOT, Asymptote, and JSON) were discussed in the section 1.3. Since this project focuses on animating string diagrams, the usefulness of the system depends on the formats currently used for string diagrams. Therefore, TikZ was chosen as the accepted input format because it is widely used for string diagrams. The other main advantage of TikZ discussed (it can be used in LaTeX documents) is likely part of the reason for its popularity. An additional advantage of this decision is that the system could be easily extended to allow for the DOT format using the `dot2tex` package [Fauske, 2006]. Example TikZ code is shown below for a diagram with two nodes and a line connecting the nodes.

```
\node [style=red dot] (0) at (-1, 0) {$a$};
\node [style=red dot] (1) at (1, 1) {$b$};
\draw (0) to (1);
```

After the input format had been decided, the parsing method needed to be considered. One possibility for parsing was to generate a lexer and parser for TikZ using lexer and parser generators such as Lex [Lesk and Schmidt, 2012] and Yacc [Johnson, 2013]. Both of these could be integrated into the project using a package such as PLY (Python Lex-Yacc) [Beazley, 2020]. Generating a lexer and parser would allow a wide variety of TikZ to be used in the system input.

However, the TikZ used for string diagrams is limited and has a regular structure, which allows for a simpler method of parsing using regular expressions. Using regular expressions, the information describing nodes and lines can be extracted from the TikZ text by searching for the node and line declarations in each TikZ diagram. While this may not be scalable for diagrams with more than only nodes and labels, generating lexers and parsers would unnecessarily complex for a limited domain such as string diagrams.

2.1.2 TikZ Types

This project mainly considered two formats of TikZ: the TikZ output from the TikZiT and FreeTikz tools. The parsing of both of these formats was implemented. Using the output of these tools makes the system particularly useful to users of these tools, as they will not be required to change the format of their TikZ diagrams. It also restricts the TikZ to regular structures which are feasible for parsing with regular expressions. This subsection describes the structure of these two formats and discusses their differences. When compiling TikZ code from either tool, a `.sty` file containing settings is also required, but this file is not needed for this system.

2.1.2.1 TikZiT

TikZiT uses a style file to specify all of the node styles to be used in a diagram. Each style in this file consists of the style name and the associated properties. The example below defines a new style called `white circle` which is a circle shape with a fill colour of white and a outline colour of black.

```
\tikzstyle{white circle}=[fill=white, draw=black, shape=circle]
```

Each TikZiT diagram consists of nodes and lines (`draw` in TikZ). For each node, the `\node` declaration is followed by a style name (from the style file), ID number, location coordinates, and label. The label is treated as LaTeX, so can be placed inside `$` signs to use mathematical notation. The example below creates a node with the `white circle` style properties and an ID of 0, located at the coordinates `(-7.25, 0)` with the label `a` written using the mathematical LaTeX font as a.

```
\node [style=white circle] (0) at (-7.25, 0) {$a$};
```

For each line, the `\draw` declaration is followed first by three optional properties: `in`, `out`, and `looseness`. The property `in` specifies the angle at which the line enters the second node and `out` specifies the angle at which the line exits the first node, where the angle starts from the left at -180 degrees and increases anticlockwise up to 180 degrees. The property `looseness` specifies how "loose" the line appears (imagining it as a piece of string), where a `looseness` of 0 is a straight line (i.e. completely tight) and the default is 1. As shown in figure 2.2, a higher looseness makes the line appear more "wavy". The next two numbers specify the IDs of the start and end nodes. An additional `.center` may be added to the ID number, but this has no effect as the line is already drawn between the centres of the given nodes. The example below creates a line between the nodes with IDs 5 and 2, and specifies that the line should enter node 2 from the left, leave node 5 from the right, and should appear slightly tighter than the default.

```
\draw [in=-180, out=0, looseness=0.75] (5.center) to (2);
```

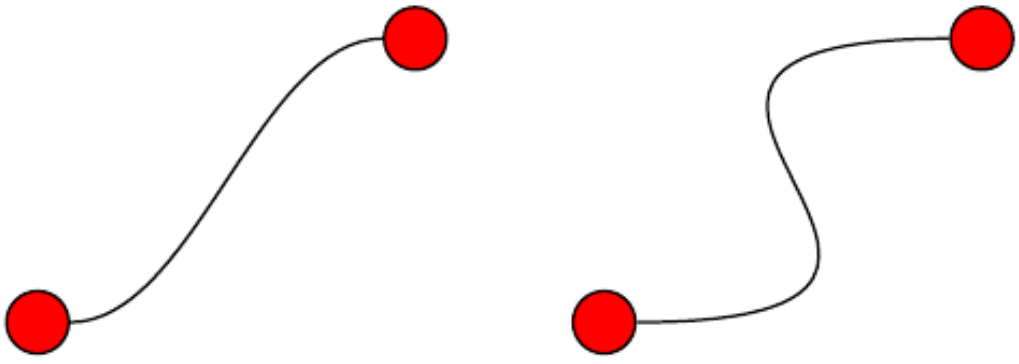


Figure 2.2: Effect of looseness.
Left: `\draw [in=180, out=0, looseness=0.75] (0) to (1);`
Right: `\draw [in=180, out=0, looseness=2.75] (2) to (3);`

A possible alternative to the `in` and `out` angles is the `bend left` or `bend right` angle. Each of these takes a single angle from 0 to 180 degrees. Looking at the left-most node of two nodes connected with this line, the angle increases anticlockwise when using `bend left` and clockwise when using `bend right`. Any line using `bend left` or `bend right` can be written as an equivalent line using the `in` and `out` angles (e.g. `[bend left=45]` is equivalent to `[in=135, out=45]`). The example below creates a line between the nodes with IDs 0 and 1, and specifies the line should leave node 0 in a north-west direction and enter line 1 from a north-east direction. Examples diagrams using `bend left` and `bend right` are shown in figure 2.3.

```
\draw [bend left=135, looseness=1.5] (0) to (1);
```

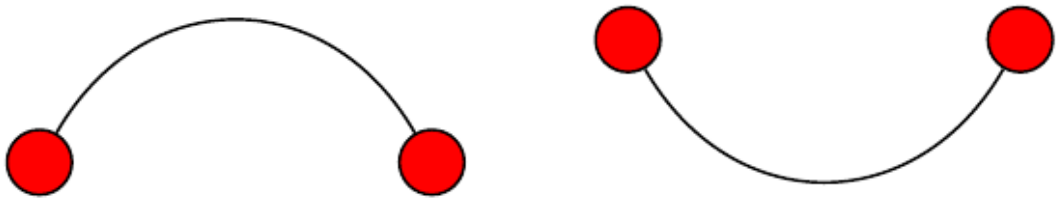


Figure 2.3: `bend left` and `bend right`
Left: `\draw [bend left=60, looseness=1.25] (0) to (1);`
Right: `\draw [bend right=60, looseness=1.25] (2) to (3);`

2.1.2.2 FreeTikz

TikZiT is designed for the creation any diagrams which can be built using simple nodes and lines, with styles to add attributes such as colour or to change the shape and size of nodes. FreeTikz is designed specifically for facilitating the drawing of graphical languages for monoidal categories, so shapes which are most common in these graphical languages (such as morphisms and dots) are particularly easy to draw in FreeTikz.

One major difference between FreeTikz and TikZiT is that FreeTikz does not use a styles file. Instead, all styles (such as `morphism` and `dot`) are defined in the `.sty` file. Therefore, there is no need to parse various styles and properties when creating an animation. This approach to styles is also the only difference between the nodes of TikZiT and FreeTikz, as the FreeTikz nodes simply state the style name within square brackets without a `style` attribute name. The example below creates a node with the morphism style and an ID of `m0`, located at the coordinates `(-7.25, 0)` with the label `m0`.

```
\node[morphism] (m0) at (-7.25, 0) {m0};
```

For each FreeTikz line, the `\draw` declaration is followed by the start location, the angles at which the line enters the second node and exits the first node, and the end location. Locations of line endpoints are not restricted to node IDs and can be written as coordinates. Compass directions (e.g. `.south east`) can also be added after a node ID to specify a specific point on the edge of a node as a line endpoint (e.g. `.south east` refers to the point which is in the south-east direction from the node centre). This

feature makes the FreeTikz format considerably more difficult to implement than the TikZiT format (see section 2.2.3.4), so the TikZiT format was implemented first in this project. The example below creates a line from the south-east edge of the node with ID `m0` to the location with coordinates `(8.25, 2)` and specifies that the line should leave the start location at an angle of -60 degrees and reach the end location at an angle of 90 degrees.

```
\draw (m0.south east) to[out=-60, in=90] (8.25, 2);
```

2.1.3 How it works

The parser takes TikZ text as input. It is designed to be run once per diagram, so the TikZ text should define a single diagram. If the TikZ is in the TikZiT format, the style definitions should also be taken as input. However, as discussed above, the style definitions are not required for the FreeTikz format.

Parsing the TikZ is carried out using regular expressions. Because of the regular and restricted structure produced by the TikZiT and FreeTikz tools, regular expressions can easily be used to extract all included information. The only values which need to be converted during the parsing phase are `bend left` and `bend right` angles, which are converted into the equivalent `in` and `out` angles using the rules below. The letter b denotes the value of the `bend left` or `bend right` angle.

$$bend\ left = b \quad \rightarrow \quad out = b, \quad in = 180 - b$$

$$bend\ right = b \quad \rightarrow \quad out = -b, \quad in = -180 + b$$

Python objects are created for each line and node and are used to store the information extracted using regular expressions. The output of the parser is a single Python object which represents the entire diagram and contains all nodes, lines, and styles.

2.2 Converting Values

2.2.1 Modularity

Modularity of the system was a major factor taken into account in the design stage. This was the reason for the addition of a separate section to deal with converting all TikZ values into values for animation with Manim (reasons for choosing Manim are discussed in section 2.3.1). This modularity ensures that sections of the system are as loosely coupled as possible, which facilitates future scaling or extending the system.

Multiple future extensions were considered during the design of the system:

- The addition of TikZ formats is the simplest extension facilitated by the modularity of the system. If the output of another TikZ tool is widely used, regular expressions could again be used to extract important information. Additionally, the `dot2tex` package could be used to extend the parser to allow for DOT diagrams. The only requirement is that the information given to the value converter uses Python objects in the same format.

- As discussed in the parsing section, using a lexer and parser generator instead of regular expressions could increase the variety of TikZ which the system can accept. TikZ is a general graphical programming language and is not restricted only to simple string diagrams, so using a more complex parser could significantly increase the control available to users while creating animations. However, features would still need to be added into the value converter and animation sections in order to display new properties and shapes accepted by the parser.
- Separation of the parser and animation also allows the system to be used with a different animation package, such as those discussed in the background section. Therefore, another animation package could be used if Manim is unable to produce the types of animation required by some users or the advantages of other animation packages are particularly appealing.

2.2.2 Input and Output

The value converter takes the output of the parser as input. As discussed above, this is a Python object which represents a single diagram and contains all information extracted from the TikZ about nodes, lines, and styles. The value converter then converts all values stored in the object to values which can be used by Manim (e.g. converting coordinates). Finally, it outputs a single Python object which represents a diagram and contains a list of node objects and line objects. Note that it does not contain styles, as Manim treats these attributes as node properties so this information is stored in the node objects. Figure 2.4 illustrates the full process from a TikZ diagram to a Manim diagram, including the intermediate representations passed between the main sections of the system.

2.2.3 Value Conversion

2.2.3.1 Property names and colours

The simplest values to convert between TikZ and Manim are the property names. For example, the colour of a node outline is specified with the property `draw` in TikZ and the property `stroke_color` in Manim. Since there is a limited number of property names, they can be converted by simply hard-coding which TikZ property maps to which Manim property. However, to give the user more control over the style of the animation, properties which do not match the accepted list are passed directly to Manim. Therefore, users have the option to make use of Manim's properties if the list of converted properties is insufficient.

A second group of values which can be easily converted is colours. The number of possible colours is larger than the number of property names, but it is still limited, so the same solution would be possible. However, Manim offers a function to parse a colour from text and return one of Manim's colours. Making use of this function allows for a large number of colours without needing a long and error-prone list. Colours which cannot be parsed can then be added manually.

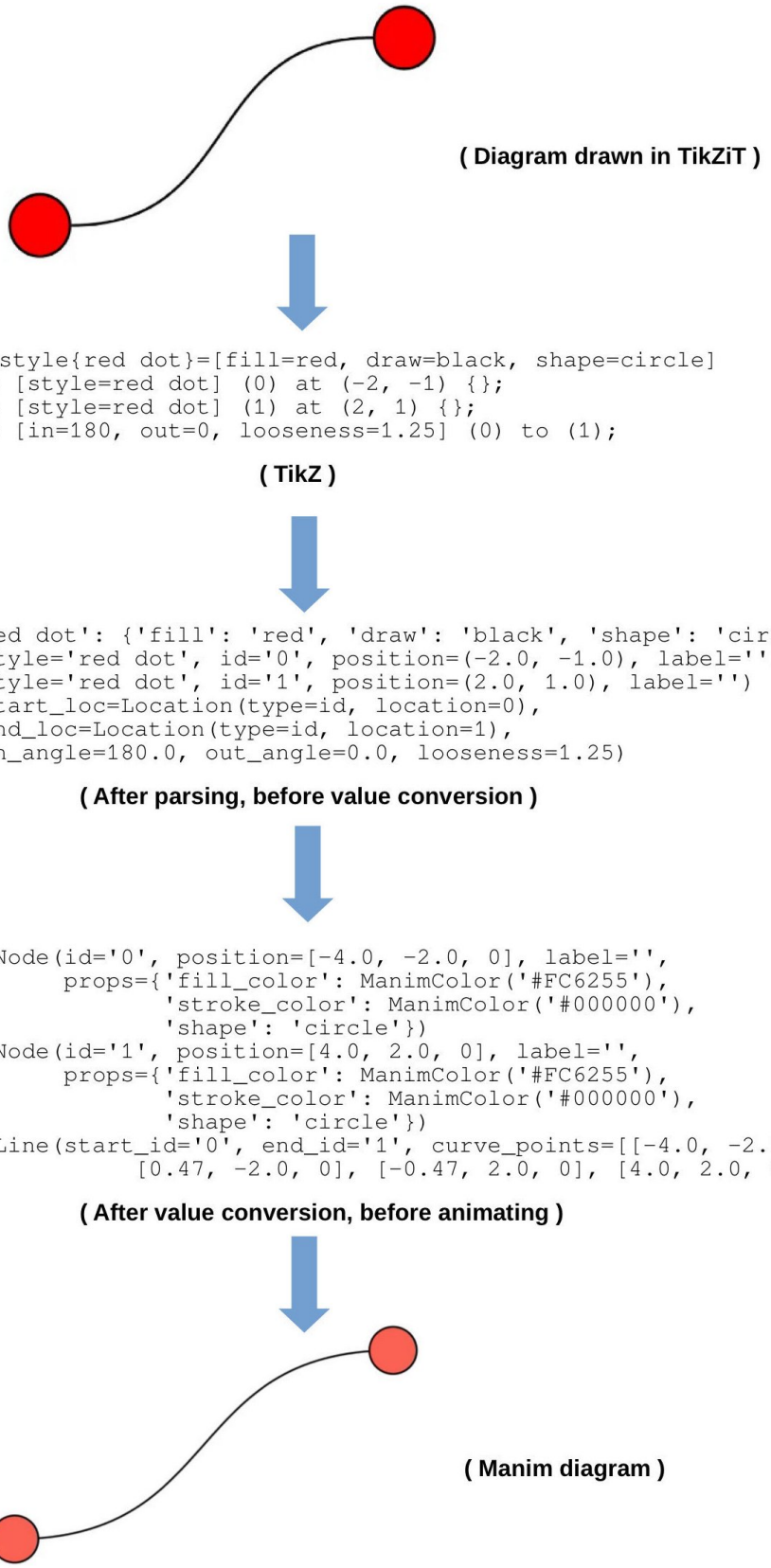


Figure 2.4: Process and intermediate representations from a TikZ diagram to a Manim diagram

2.2.3.2 Coordinates

Coordinates also need to be converted from TikZ to Manim. TikZ coordinates are not limited to a specific range for each diagram, and, therefore, TikZ diagrams are scaled to fit into documents. In contrast, Manim coordinates range from about -7 to 7 on the Y-axis and from -4 to 4 on the X-axis (although it is possible to configure this).

The original method to convert coordinates dealt with each diagram individually because the value converter deals with one diagram at a time. It calculated the minimum and maximum X and Y coordinates in the given diagram and scaled every coordinate relative to these values. However, if these minimum and maximum values changed between diagrams in a single animation, the entire coordinate system would change. For example, if the first diagram consists of only node 1 at the TikZ coordinates (0,0) and node 2 at (1,0), and node 1 moves to (2,0) for the second diagram, the resulting animation would start with one node on either edge of the screen, which would then switch places with each other. This is shown in figure 2.5. Therefore, it was necessary to calculate the minimum and maximum X and Y coordinates between all diagrams in the animation and use these values as the limits when converting coordinates within each diagram.

Note that there may still be bugs present in calculations involved, especially for diagrams with a small number of nodes. However, a user can easily fix issues with these calculations by adding four invisible nodes (i.e. one node with `style=none` on each edge of the diagram) to help the system find the minimum and maximum coordinates.

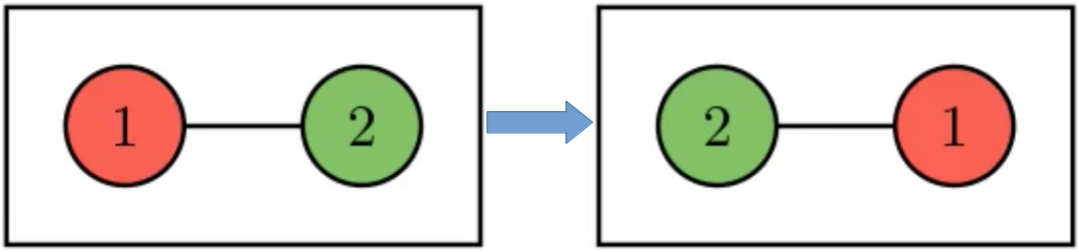


Figure 2.5: Two nodes switching places in the same animation without switching coordinates.
Left: Nodes 1 and 2 have coordinates (0,0) and (1,0) respectively.
Right: Nodes 1 and 2 have coordinates (2,0) and (1,0) respectively.

2.2.3.3 Lines

Converting the line representations from TikZ to Manim was particularly challenging. TikZ represents a line between two points as an `in` angle, an `out` angle, and a `looseness`. In contrast, Manim represents a line as four handle points of a Bézier curve. The line endpoints were given in the TikZ representation, but the positions of the two inner handle points needed to be calculated using the following method. The angle between each endpoint and their closest inner handle point was given by the `in` and `out` angles. The distance between each endpoint and their closest inner handle point could be calculated using the following fact: if d is the distance between the two

endpoints, a looseness of 1 indicated that the inner handle points were $\frac{2}{5}$ of the distance d from their closest endpoint, and this distance is scaled with the value of looseness. For example, if the distance between the two endpoints is 5 (in TikZ units), a looseness of 1 specifies that the distance between each handle point and its closest endpoint is 2 TikZ units, while a looseness of 2 would scale this value to 4 TikZ units. Since lines which were specified in TikZ using `bend left` or `bend right` were converted to `in` and `out` angles in the parsing section, this does not need to be considered here.

2.2.3.4 FreeTikz compass directions

The use of compass directions to specify a point on the edge of a node (e.g. `m0.south east`) was the most challenging aspect of FreeTikz syntax to implement. This was because Manim does not use compass directions and the TikZ specification of nodes only provides the size and shape of the node and the location of the node's centre. Therefore, the location of a point on the edge of a node needs to be calculated from this information, which is particularly challenging for asymmetric shapes such as the FreeTikz morphism shape. The conversion method is briefly outlined below. First, the compass direction is converted to a direction in Manim (e.g. `south east` becomes $DOWN + RIGHT$). A line is then created from the node centre in this direction, using the shape size to ensure the line is long enough to intersect the shape outline. Manim is then used to get a series of points which would define the shape outline (this is how Manim represents shapes internally). Finally, the system creates lines connecting these outline points and finds the intersection of any of these lines with the line from the centre. This method ensures that the correct point can be found on any shape, including asymmetric shapes such as the FreeTikz morphism, without rewriting the calculations for each shape.

2.3 Animating

2.3.1 Manim

Manim was chosen as the animation tool for this project. The main reason for this decision is that the other main candidates (e.g. GSAP and D3.js) are designed for addding animation to websites, not for producing a video file of an animated diagram. Manim is very powerful for simple mathematical animations and automatically produces a video file when an animation is created. It can provide an image of the final frame without needing to render the entire animation, so users of this application could be given a quick preview. Manim also offers useful configuration options for quality and background colour to give users more control over the animation style.

2.3.2 Creating diagrams

Each string diagram in an animation first needs to be represented using Manim. Manim works by creating objects (called Mobjects) using Manim's classes (e.g. `Circle` or `CubicBezier`) and adding these objects to a `Scene` object. The `construct` method of the `Scene` is where these objects are added to the `Scene` and any movements or changes

to be made as part of the animation are specified. All nodes and lines in the string diagram need to be created, added, and animated in this way.

The system needs to produce a variety of different types of node, so a general `Node` class was written. Constructing an object from this class requires a Manim shape object, any text for a label on the node, and any other properties such as size or colour. Each type of node then uses and builds upon this general `Node` class to create more specific nodes. The system can currently create squares, rectangles, circles, dots, and the non-symmetric morphisms from FreeTikz, but this design intentionally allows for the easy addition of other types of node, such as new shapes. In order to facilitate the automatic creation of nodes from the value converter output, extra functionality was added for creating nodes given only text specifying the type of shape (e.g. "circle").

Manim allows grouping of Mobjects, so a `Diagram` class was written to group all nodes and lines for a given diagram into one Mobject. String diagrams produced using TikZiT display nodes on top of lines, so this system splits each diagram into a node layer and a line layer to have the same effect. When creating a new diagram, all lines and nodes are created and stored within the `Diagram` Mobject, and made accessible using their IDs. Only nodes have an ID given in the TikZ diagram, so the start and end points together were treated as line IDs. However, this will cause problems if a diagram contains two lines which have the same start and end points. When this causes problems, one possible solution is to allow users to give lines optional IDs in the TikZ code in a format similar to TikZ node IDs. The system would use this given value if it is present or use the start and end points if this given value is not present.

Example TikZ code and the corresponding Manim diagram is shown in figure 2.6.

```
\node [style=none] (0) at (-5, 0) {};
\node [style=red dot] (1) at (-2, 0) {$a$};
\node [morphism] (2) at (2, -2) {$b$};
\node [dot] (3) at (3, 2) {};
\node [style=none] (4) at (5, 0) {};
\node [style=none] (5) at (0, 1.5) {$\beta$};
\draw (0) to (1);
\draw [bend right=60, looseness=1.25] (1) to (2);
\draw [in=-180, out=0, looseness=1.5] (2) to (4);
\draw (1.north east east) to[out=20, in=-180] (3);
```

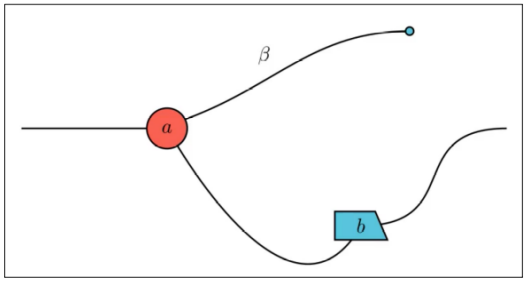


Figure 2.6: TikZ code and the corresponding Manim diagram, using the style: `red dot=[fill=red, draw=black, shape=circle]`

2.3.3 Animation

The easiest solution to animate between a series of string diagrams would be to construct a series of `Diagram` Mobjects and simply tell Manim to transition between them. Since the diagrams are represented as Manim's Mobjects, Manim can smoothly transition between them, changing the locations and any properties of the contained nodes and lines. However, as soon as a node or line is added or removed in any diagram, Manim completely fails to map nodes and lines in one diagram to nodes and lines in the next.

The method chosen for this system deals with nodes and lines separately and aims to map nodes and lines from one diagram to nodes and lines in the next. For each valid mapping it finds, it makes use of Manim's available functionality to animate the transition. Manim's `FadeOut` functionality is used for nodes and lines which are in one diagram but not in the next, and Manim's `FadeIn` functionality is used for nodes which are not in one diagram but are in the next. Lines which are not in one diagram but are in the next are added using Manim's `Create` functionality which draws the line from its start point to end point.

Node transitions are the simplest to deal with. If there is a node with the same ID in two consecutive diagrams, Manim's transition functionality can animate any change between the nodes. If one diagram contains a node with an ID which is not present in the next diagram, the node fades out. Conversely, if a node ID is not present in one diagram but appears in the next diagram, the node fades in.

Line transitions are more difficult because lines do not have such simple IDs. Since the start and end points are treated as the ID of a line, if a line in one diagram has the same start and end points as a line in the next diagram, Manim's transition functionality animates any changes (e.g. a change in looseness). Instead of making all other unmatched lines appear or disappear, if there are unmatched lines in the first diagram with the same start point as unmatched lines in the second diagram, transitions are created between these lines. Then, the same step is taken for unmatched lines with the same end point. After this process, any remaining unmatched lines either fade out or are drawn in. This process makes the assumption that a user is more likely to want to animate changing the end point of a line than the start point, but may still want to animate changing the start point.

Chapter 3

Implementation

3.1 Web Hosting

3.1.1 Command Line

The first step to allow users to create their own animations was to allow access through the command line. This was easy to implement and allowed users to create animations by specifying the file path for each TikZ diagram. This becomes easy to use with practice, but the user must understand how to use the command line and each TikZ diagram must be saved in a separate text file. In addition, the command line flags for any Manim configuration required (e.g. video quality or background colour) would need to be memorised by the user. However, this would provide the user with more control over the configuration.

3.1.2 Making the application available

The easiest method to make the application available would be to make a public GitHub repository. This would allow users to download the code and run it on their own computers. Users would need to install required packages (such as Manim) themselves before being able to run the code. Installing these packages could fail or result in unexpected errors, so users may need some technical knowledge to solve these problems.

Another option is to make pre-build versions of the software available for users to download, which is the approach taken by TikZiT. This is more difficult to develop and a different version may be needed for each operating system, but it is likely to be easier to set up for users. In many cases, users would be able to use the software without technical difficulties, but some errors may still occur.

With both of these options, users may require help to deal with errors or they may give up if no help is available. Unless we accept the disadvantages of a command line interface, a graphical user interface (GUI) would also be required, which increases the chance of errors.

The approach taken in this project is to host the software online. The main advantage of this is user convenience, as the user would simply be able to visit the website without needing to download any software. This also removes the need for any technical knowledge, as a GUI would be provided on the website. The main disadvantage is that the system would require a server to run on and this could be costly. Hosting software can also be particularly challenging when many dependencies are required (as shown in section 3.1.5.3).

Whether downloaded or hosted online, a GUI provides a few additional advantages. First, it allows the developer to provide convenient configuration options, removing the need for users to understand and remember command line flags. Second, a GUI allows a preview image of the animation's final frame to be displayed so the user does not need to wait for a full render to be completed for each small change. Users can produce a preview image using the command line, but the image would be stored on their computer and the user would need to find this image to display it. Finally, an online GUI could display a preview of the entire animation without storing the video file and any additional cached files on the user's computer.

3.1.3 Back-end

Any of a range of different languages could have been chosen for the system's back-end, but Python was chosen to be consistent with Manim and the rest of the system. This gave two main options for back-end frameworks: Django and Flask.

Django [Django Software Foundation, 2024] is a Model-View-Template (MVT) framework, so it is able to deal with database interactions and can be used to return entire HTML pages to the user. It is a "batteries-included" framework, meaning it includes a range of additional functionality which can be easily integrated into a project, such as authentication, built-in admin, and session management. This functionality could be implemented in future versions of the system, but the main implementation of this project only requires a basic API which can be called to run the animation program. This can be implemented with the Django REST Framework, which is installed as an additional package.

Flask [Pallets, 2024] is not a "batteries-included" framework, which makes it more lightweight and flexible than Django. Therefore, the developer needs to install extensions when additional functionality is required. When using Flask to also deal with the front-end, one possible disadvantage is that it can only be used with single-page applications. However, Flask can be used to implement a REST API, which would be the main requirement for this project.

Both Django and Flask would be suitable for this project and the choice is unlikely to have a significant effect because it will only be used as a REST API to run the animation program. In this case, Django was chosen because of the additional functionality included through being a "batteries-included" framework. However, the modularity gained through separating the front-end and back-end would allow a future developer to easily switch to Flask. They may want to do this to reduce the application size, as a REST API may not require a large "batteries-included" framework.

3.1.4 Front-end

There are many options for front-end technologies. The simplest option would be plain HTML, CSS, and JavaScript, but there are also many front-end frameworks and libraries which provide additional functionality to speed up the development process. This section discusses the advantages and disadvantages of the three most popular front-end frameworks: React, Angular, and Vue. Technically, React is a library, but we will refer to all three as frameworks here for simplicity and because the distinction between frameworks and libraries is not important for this comparison.

React [Meta, 2022] is the most common and widely-used of all three frameworks, which has some important advantages. First, there is a large community of React developers and plenty of support online. Second, it is more likely that any future developers will already know a popular framework, so they will not need to learn an additional framework before being able to work on the project. Third, React is updated often to meet the needs of the large community of developers who use it. Finally, many third-party extensions have been developed to add specific functionality to React. For example, Next.js can be used for server-side rendering and Redux can be used for managing state in applications. In addition to popularity, React has a couple of other advantages: React is fast because of its use of a virtual document object model (DOM), and React can also be used in React Native to develop mobile apps, without the need to learn a new language or framework.

Angular [Google, 2023] is the second most widely used framework. It is designed for building large, testable and manageable applications. One area in which it stands out from other frameworks is its use of dependency injection, which allows the efficient handling of complexity in large projects. However, Novac et al. [2021] mentions a few of Angular's disadvantages. First, the framework's large size results in slower loading times. Second, Angular is complex and is designed for large projects in large companies, so it is generally more difficult to learn than other frameworks. Finally, Angular uses TypeScript, so developers need to learn an additional language before using it.

Vue.js [You, 2024] is less widely used than the other two, but still performs better in some aspects [Kaluza et al., 2018]. Vue is the fastest out of all three because it uses a virtual DOM and it has a small size, allowing faster loading times. Using Vue, developers can add functionality incrementally, which makes the framework more flexible and maintains a small size. In addition, it has clear and concise documentation, which can make a significant difference in development. However, the fact that Vue is less widely used than the other two frameworks reduces the chance of future developers already having experience with it. The smaller community also results in a smaller number of third-party extensions [Novac et al., 2021].

For this project, React was chosen for the front-end. The main reasons for this decision were React's popularity and the resulting advantages, such as online support, frequent updates, third-party extensions, and the higher chance that future developers can already use it. In addition, if a mobile app was required, transitioning to React Native from React would be easier than from other frameworks. Angular is designed for large projects and is unnecessarily complex for this project. Vue has some useful advantages, but lacks the popularity and related benefits of React.

3.1.5 Deployment

3.1.5.1 Back-end deployment options

Two of the most popular free options for hosting a back-end application are Render and Vercel. They can deploy an application from a git repository and automatically run a build command each time an update is pushed to the repository. They also use their content distribution networks (CDN) to move websites and content closer to users to reduce loading times. However, an important disadvantage of Render is the fact that the virtual server instances which host back-end apps spin down with inactivity, so requests can be delayed by about one minute. Therefore, Vercel is a better choice than Render when hosting a back-end app for free.

Despite not being free, it is also important to consider the main cloud service providers. AWS, Google Cloud Platform, and Microsoft Azure are all reliable, scalable, and offer a range of services which can be easily integrated into a project. The most popular of these is AWS, which offers services such as App Runner and Elastic Beanstalk to allow developers to deploy applications without needing to deal with the underlying infrastructure or virtual machines. AWS is so widely used that most developers will likely have experience working with it, which is an important advantage over the free hosting options.

3.1.5.2 Front-end deployment options

There are many options for deploying front-end applications, however, there are a couple of aspects which need to be considered when choosing a front-end hosting platform. First, the cost can vary between hosting platforms, but many options are either completely free or free for a trial period. Second, many free options do not include SSL/TLS certificates for HTTPS, which can cause problems on some browsers or make users anxious, as network traffic is left unencrypted. Storage options, such as AWS S3 and Google Cloud Storage, can host static sites but often do not include SSL/TLS certificates. This section discusses some options which are free and include SSL/TLS certificates.

The first options to consider are companies who focus on hosting websites: Render, Vercel, and Netlify. These have the same advantages as those discussed for Render and Vercel in the previous section: they offer automatic deployments from GitHub and provide access to their CDNs. Fortunately, front-end applications do not spin down with inactivity on Render, so there are no delays when accessing a front-end application hosted on Render.

Some additional options include GitHub Pages and AWS Amplify. GitHub Pages is generally the simplest option for deploying a simple web app, but it does not automatically run a build command on each update to the repository. Therefore, it may require more effort than the first three options when updating a project. AWS Amplify offers access to the large CloudFront CDN and easy integration with other AWS services, but can only be used to host an application for free for the first 12 months.

3.1.5.3 Deploying the project

This section describes the deployment of this project on multiple platforms. The back-end was particularly challenging to deploy, so the back-end deployment was attempted on four platforms: Vercel, Render, AWS App Runner, and AWS Elastic Beanstalk. The front-end deployment was relatively simple, so was only deployed on Render.

Vercel is a particularly strong candidate for back-end deployment because it is free and avoids Render's delays, so the project was deployed on Vercel first. Vercel requires little configuration: only a `build_files.sh` file with a couple of shell commands (e.g. to install packages), a `vercel.json` file, and a few changes to the Django settings file such as removing the default SQLite3 database configuration. Instructions for this configuration can be found easily online. One disadvantage of Vercel is that it currently only supports Python 3.9, so Django also needed to be downgraded from version 5.0 to 4.2 to be compatible. However, the main errors which occurred while deploying to Vercel were related to Manim's dependencies, which was a disadvantage of Manim discussed in section 1.4.2. First, Vercel was unable to install the `ManimPango` dependency directly using `pip` because it could not find the package `pangocairo` which needed to be installed first. Despite the fact that Vercel abstracts away the underlying infrastructure and operating system, solving this issue required the knowledge that Vercel uses AWS and the Amazon Linux 2 operating system, which is RPM-based and uses an RPM-based package manager. Therefore, the package management tools `yum` and `dnf` needed to be used instead of a Debian-based tool such as `apt`. Second, adding Manim to the project caused the error "A Serverless Function has exceeded the unzipped maximum size of 250 MB", which may be because of the large number of dependencies Manim requires. Since it was not possible to remove Manim or any other files from the project and Vercel's logs were not detailed enough to check for other causes of the error, this suggested that Vercel was not suitable for deploying the back-end.

The second platform used to deploy the back-end was Render. This was even easier than Vercel, as it required almost no configuration and no errors occurred during the build stage. However, the application produced a status 400 error for all requests once it was online. Errors such as this are particularly difficult to debug in Render because the logs give no information about this type of error and Render requires a paid plan to access the shell for the virtual server. For this reason and the delays discussed previously, Render is not suitable for deploying the back-end of this application.

AWS App Runner is the AWS version of Vercel and Render. It includes the additional advantages of AWS discussed previously (reliable, scalable, popular), but is not included within the AWS free tier one-year trial period and can be expensive. Like Vercel and Render, it offers auomatic deployments from a GitHub repository, but this further increases the cost. Deploying to App Runner was initially as convenient as Vercel and Render, but a single error in deployment caused all following deployments to rollback to the previous working deployment. Since this appears to be a bug in AWS App Runner rather than an issue which can be fixed in the project, App Runner is also unsuitable for deploying the back-end of this project.

Finally, the back-end was deployed on AWS Elastic Beanstalk, which appears to be the most suitable option for this project. Elastic Beanstalk uses AWS EC2 virtual machines and AWS S3 storage, abstracting away some of the underlying details, but not to the same extent as Vercel, Render, and AWS App Runner. There is only a charge for the underlying services used (EC2, S3), not for Elastic Beanstalk itself. Since basic usage of EC2 and S3 is free for the first year, this deployment method is almost free for the first year and then only requires payment for the services used, without a standing charge.

The main difference between Elastic Beanstalk and the previous services is the increase in both control and the required configuration. Deploying on Elastic Beanstalk is more complex than Vercel, Render, or App Runner, but errors are easier to debug as it is possible to configure custom logs. There is also more guidance and documentation online because Elastic Beanstalk is a popular service. However, even with online documentation, configuration can be difficult. For example, during deployment it was necessary to use the AWS Identity and Access Management (IAM) service to create an identity profile role. In addition, Elastic Beanstalk configuration requires working with YAML. One advantage of Elastic Beanstalk's YAML configuration files is the availability of multiple package management tools, as some of Manim's dependencies are easier to install using `dnf` than `yum`. However, `ffmpeg` (one of Manim's dependencies) required a complex YAML script, as it was not available through `dnf` or `yum`. This is another example of the issues caused by Manim's dependencies.

In contrast to back-end deployment, deploying the front-end was easy and convenient. Render was chosen to deploy the front-end as it is free, requires the least configuration, and Render does not add delays to front-end applications. However, deploying the front-end exposed an additional problem with the back-end: Elastic Beanstalk does not automatically provide an SSL/TLS certificate for HTTPS, which results in browsers blocking responses from the back-end with the error "Blocked loading mixed active content". Mixed content refers to content which is transferred using both the HTTP and HTTPS protocols, and this is blocked for security. Therefore, the back-end required additional configuration to fix this issue. First, it was necessary to register a domain name. Second, this domain name needed to be configured on the AWS Route53 service to direct to the Elastic Beanstalk application. Then an SSL certificate needed to be generated using AWS Certificate Manager (ACM). Finally, an AWS application load balancer (ALB) needed to be created, as this is the easiest way to associate an ACM certificate with an Elastic Beanstalk application. Fortunately, creating a certificate with ACM is free when integrated into an ALB.

3.2 Extra features already implemented

The first feature which may be useful for users is the ability to replace an entire section of a diagram with a new section. In fact, this is possible with the original implementation and does not require additional code. To replace one section with another, the TikZ code can simply be adjusted to reflect the required change. As long as the IDs of the nodes and lines are not the same in the two sections, the nodes and lines in the first section will fade out and the nodes and lines in the second section will fade in. However, some

users may need to emphasise the fact that an entire section is replacing another (e.g. surrounding the section with a box as it fades in and out). This would also be possible with the current functionality, but it may be helpful to include an additional feature to facilitate this process.

For various reasons, some users may need to add short subtitles to their animations. Since string diagrams are mathematical diagrams, this could be to specify the rule or law which justifies the change from one diagram to the next. To allow users to add these subtitles, the front-end has an additional input for each TikZ diagram input. If the user enters text in one of these inputs, the text will fade in as a subtitle during the transition to the corresponding diagram and fade out during the transition to the next diagram. While the use of Manim makes the addition of a subtitle relatively easy, one complication was the fact that the size of the diagrams in an animation need to be slightly smaller to make space for subtitles if any of the diagrams in that animation contain a subtitle. Therefore, this also affected the conversion of coordinates from TikZ to Manim in the value conversion section.

The original implementation forced the user to download an animation to see it, but this can result in the user needing to download many video files before they are satisfied with the animation. Therefore, a video preview was added to the front-end, allowing the user to pause the animation or use a slider to manually move through the animation frames without the need to download a video file first. When the user is satisfied with the animation, they can click a button to download the video file.

Chapter 4

Evaluation

4.1 Usability Study

Up to this point, design choices have been based on information and research available online. However, when combining multiple design choices into a single specific tool, it is important to test the choices by interacting with potential users. A usability study was designed to gather knowledge and advice from potential users. It aimed to discover if this tool suits their needs and which aspects can be improved in the future.

Only potential users who have experience with string diagrams and graphical programming languages were suitable subjects for the usability study. This made the task of finding subjects particularly challenging, so only a small number of potential users were found and six of these agreed to participate. A larger number of participants would have been more reliable and would have provided more ideas for improvements to the tool, but it was still possible to gather useful insights by qualitatively analysing the responses from six participants.

The usability study consisted of five questions: three before trying the tool and two after.

4.1.1 Before trying the tool

The first three questions aimed to gather information about the type of tool which would be most useful to potential users. These questions were asked before participants saw the tool to ensure their answers would not be affected by aspects of this project. At this point, the participants were only told the following:

> *The tool is designed to take a series of string diagrams as input, where each diagram is specified in some text-based format (e.g. a graphical programming language). The tool animates the steps between these diagrams.*

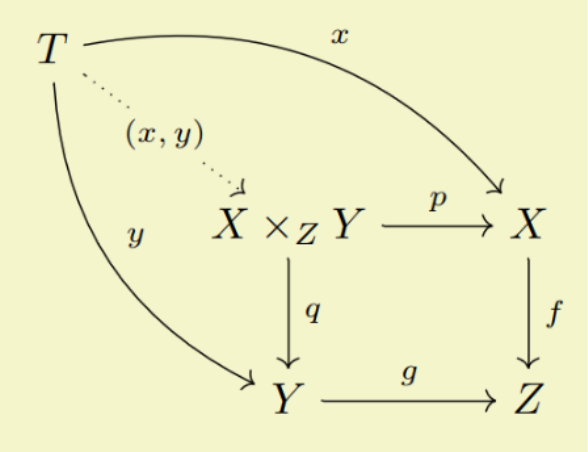
4.1.1.1 Question 1: Input formats

> *Which input format for diagrams would you find most useful and why? This could be the format you currently use to draw diagrams (in papers etc.).*

Section 1.3 discussed possible input formats for this project (TikZ, DOT, Asymptote, and JSON) and section 2.1.1 explained why TikZ was chosen. The main advantages of TikZ over the other options were its popularity and the fact that it can be embedded in LaTeX documents. This question was used to discover whether TikZ is the most useful format for potential users or if the tool should accept other input formats.

The responses from four out of six participants support this decision. Two participants wrote that TikZ would be a useful input format for the tool and the need for TikZ compatibility was even reiterated in question 3. An additional two participants mentioned that they usually draw string diagrams using TikZiT. This supports the decision to focus on the subset of TikZ used by TikZiT.

One participant mentioned `tikz-cd`, which is an extension of TikZ designed to facilitate the drawing of commutative diagrams [Neves, 2011] (an example is shown in figure 4.1). The convenient syntax provided by `tikz-cd` is not currently accepted in this project, but this could be a useful feature to add to future versions of the tool. To allow `tikz-cd` syntax, the sections of the system which deal with parsing TikZ and converting TikZ values to Manim values could be extended. As shown in the figure, `tikz-cd` is designed for a set of diagrams which are slightly different from those focused on in this project, but which are similar enough that only a small number of features (e.g. arrows) would need to be added to the animation section of the system.




```
\begin{tikzcd}
  T
  \arrow[bend left]{dll}{x}
  \arrow[bend right]{ddl}{y}
  \arrow[dotted]{dl}[description]{(x,y)} & & \\
      & X \times_Z Y \arrow{l}{p} \arrow{d}{q} & X \arrow{d}{f} \\
      & Y \arrow{l}{g}                         & Z
\end{tikzcd}
```

Figure 4.1: An example of `tikz-cd` code and the resulting diagram. [Neves, 2011]

Two responses mentioned that taking images of string diagrams as input would be useful. This would require the tool to extract all information about the string diagram (nodes, lines, labels, colours etc.) from the pixels in an image. While this is possible, it would be far more complex than parsing a graphical programming language and would require techniques such as edge detection to locate shapes and text recognition to extract labels. The main functionality of FreeTikz is using image processing methods such as these to produce TikZ code from hand-drawn string diagrams, so users may be able to use FreeTikz together with this application when they require this functionality. If this functionality needs to be included within this application, there are tools which can help with image processing, such as OpenCV [Pulli et al., 2012] and Mahotas [Coelho, 2013], but including additional tools will increase the number of dependencies, which increases the chance of errors during deployment (as found in section 3.1.5.3). However, this may be a useful feature to add in future versions of the tool.

One response suggested algebraic formulae as input. One option for users who would like to animate the diagrams represented by a series of formulae would be to use another tool to produce TikZ for each equation and use this tool to produce an animation from these TikZ diagrams. As mentioned in section 1.2.2, Chyp allows an input format which is similar to equations, so may be useful for converting equations to string diagrams. However, Chyp does not produce TikZ as output, so may be difficult to use alongside this tool. Similar to allowing image input in this project, allowing algebraic formulae as input would require the addition of a complex new feature, but could be considered for future versions of the tool.

4.1.1.2 Question 2: Making the tools available

> *How would you prefer the tool to be made available and why? (e.g. download source code from GitHub, install executable program, use online on a website)*

Section 1.2.1 discussed how other tools are made available and section 3.1.2 explained why the decision was made to host this tool online. The main advantages discussed were user convenience and making it accessible for non-technical users.

The results from the usability study strongly support this decision, as five out of six participants mentioned making the tool available online on a website. The responses included an additional advantage of hosting the tool online: this makes the tool more portable so users can access it on any device, without repeatedly installing it.

Three participants responded that installing an executable program would be useful and mentioned that an executable program would be best for offline use. Finally, two participants suggested making the source code available. However, both of these participants mentioned this alongside hosting the tool online, which suggests that this option may be best in combination with online access. This supports the findings discussed in section 1.3 where source code was rarely the only method used to make the tool available.

4.1.1.3 Question 3: Useful features

> *Which features would you find useful in the tool and why? This could be additions to the animation (e.g. subtitles), something in the application interface to improve convenience, or something else.*

In addition to the input format and the method used to make the tool available, it would be helpful to know which features the tool would need in order to be useful. These features could focus on various aspects, such as details in the animations produced, additions to the interface, or useful output formats. The vast majority of the features mentioned in responses have already been considered or implemented, but a few responses provided insights into possible future development on this tool.

Multiple responses mentioned details in the animations produced. Flexibility and the ability to customise the appearance of animations appeared in the responses of two participants. This supports the choice of Manim for animation as it is a particularly powerful and flexible package. However, the tool needs to provide users with the ability to utilise Manim's flexibility, which is an area for future improvement. One feature implemented to provide users with this ability was to allow any Manim properties to be included within the user's TikZ styles. These properties are passed directly to Manim, which provides users with more control over the appearance of animations without needing a long, hard-coded list of available properties. Manim also provides a `ManimConfig` class to allow configuration using Python, so options to configure this could easily be added to the online GUI. This could be implemented in future versions of the tool to provide more control to users. An additional response mentioned support for LaTeX rendering of labels, which further supports the decision to use Manim as it can easily integrate LaTeX into any animation.

The tool interface was another focus of multiple responses. One response mentioned a good GUI, which supports the decision that a command line interface would not be suitable for this application. Two responses mentioned convenience and another mentioned a hassle-free interface. These responses emphasise the importance of user convenience, which was the main reason behind the decision to host the application online. One response suggested a timeline tool to move through the animation frames slowly. This has already been implemented (section 3.2) to allow the user to pause the animation and use a slider to manually move through the animation frames without needing to download the video file first.

One participant mentioned that the ability to embed the animations in LaTeX papers would be useful. This is not surprising, as one of the main advantages of TikZ is the ability to embed it in LaTeX papers. There are tools available to help with embedding animations in LaTeX papers, but they do not support mp4 files, which is the tool's current output format. For example, the LaTeX package `animate` [Grahn, 2023] can create animations in LaTeX papers from a series of frames. If users have a GIF animation, they can also use `ImageMagick` [ImageMagick Studio LLC] to convert a GIF to a series of images which `animate` can take as input. Therefore, this project could allow users to receive the animation as a GIF or a series of frames.

Finally, one participant hoped the tool would allow a variety of shapes and would allow

users to add labels to lines. The tool currently supports some basic shapes (cicle, square, rectangle, dot, FreeTikz morphism) and could be easily extended to support more. Since TikZiT and FreeTikz do not give labels to lines, this feature has not been implemented in this project. However, users could add an invisible node with a label beside the line for the same effect.

4.1.2 After trying the tool

After answering the first three questions, participants were given a link to the tool and were asked to try using the tool before looking at the following two questions. Example TikZ code was provided on the web page in case they did not have any TikZ diagrams ready. After trying to use the tool, the final two questions asked participants about the aspects they liked about the tool and improvements which could be made. These questions aimed to gather detailed information specific to this implementation of the project and the results will be particularly useful for future development on the project.

4.1.2.1 Question 4: Good aspects

> *What do you think is good about the tool and why?*

Some responses to this question simply reiterated the findings from the first three questions. One participant stated that accepting TikZ as input was a good choice as it is widely used and another was pleased that the application works with TikZiT. A third participant praised the fact that no installation was required and two participants stated that the interface was clear and straightforward to use. These responses further support the decisions to accept TikZ input (specifically from TikZiT) and to deploy the application online.

Some other responses focused on the output of the application. Two participants found the output animations attractive and clear and a third participant liked the continuous movement of lines. These are additional advantages of using Manim for animation: it produces animations which are attractive, clear, and have smooth transitions with little configuration. The third participant also praised the convenience of automatically inferring which lines in the a diagram correspond to which lines in the next diagram. This was a challenging feature to implement and is described in section 2.3.3.

4.1.2.2 Question 5: Improvements

> *What could be improved to make the tool more useful and why? This could be extra features, changes to current features, or anything else.*

Asking potential users about possible improvements aims to gather information on the best areas for future development. Many aspects of the application could be improved, but not all will have the same effect on users. The responses to previous questions provided ideas for some additional features, most of which were large and complex. However, asking potential users for improvements after they have seen the tool provides them with the opportunity to suggest smaller improvements which are more specific to the current implementation of this project.

Five out of six participants suggested at least one improvement concerning control over the animation. This relates closely to the flexibility and ability to customise animations which was discussed in the question 3 responses. Two participants mentioned control over movement of nodes and lines, such as the ability to specify the path along which they move. This feature could be difficult to implement, as the tool focuses on convenient, automatic animation rather than providing animation capabilities found in animation software. For this reason, the most suitable approach may be to pre-define particular types of movement.

One participant suggested the ability to vary the animation time. Currently, each diagram is shown for one second and the transitions between diagrams each take one second. Allowing the user to configure a single value to be applied for all diagrams would be simple as this could be set as a global configuration variable. Allowing a different value for each diagram and transition would be more complex but still possible.

One participant discovered a situation where the tool incorrectly matched lines between consecutive diagrams, so suggested the ability to annotate lines to avoid these mistakes. This could be implemented by allowing TikZ line declarations to include an optional ID value, ideally in a similar format to TikZ node IDs. For each line, the system could use the optional line ID if it is present and fall back to the IDs described in section 2.3.3 (start and end points) if the optional ID is not present.

Providing a wider variety of input and output options was mentioned among the responses. The responses to previous questions suggested that `tikz-cd` and images would be useful input options. No participants asked for DOT, Asymptote, or JSON as input options, which further supports the choice of TikZ input. This also suggests that DOT, Asymptote, and JSON as input options would not be useful for users of the application. Regarding output formats, responses to previous questions mentioned the ability to embed the animation in LaTeX papers, so a series of frames or a GIF as an output format option would allow users to embed the animation in LaTeX papers using the `ImageMagick` and `animate` packages. An additional advantage of Manim is that it provides options to produce the animation as a GIF or save all frames as images, so these output formats would be relatively easy to implement in this application. One response mentioned HTML and JavaScript as a possible output in order to allow users to embed the animation in a web page. A downloaded mp4 file can be added to a web page with a few simple lines of HTML, so the application could provide users with this HTML code. However, animating the diagrams with JavaScript would require changing the animation library from Manim to one of the tools discussed in section 1.4.2, such as GSAP. The modularity of the system would make this possible without the need to rewrite the entire application, but this would still require a major addition to the system.

The remaining suggestions focused on the GUI. Some were simple changes which only affect the React front-end section of the application. Participants suggested providing a separate tab for each TikZ diagram input and the ability to remove text box inputs for TikZ diagrams when they are no longer needed. They also suggested the ability to load the example TikZ by pressing a button and hide the example when it is not needed.

One response mentioned the fact that the application currently does not provide helpful error messages when a problem occurs. For example, if the user enters TikZ code which

causes the back-end of the system to raise an error (e.g. two nodes with the same ID), the preview will not load and a simple error message may appear to instruct the user to try again. To provide helpful error messages, changes would need to be made in all back-end sections (parser, value converter, Manim animation, back-end API) so that a problem could be diagnosed and returned to the front-end in place of a failed animation.

Another participant suggested providing a live preview of each diagram while a user edits the TikZ code. This feature was not previously considered because the tool was designed for users who already have the TikZ code for a series of diagrams. This would be particularly difficult to implement with the current system architecture, as Manim deals with the rendering of all diagrams in the back-end. Therefore, live previews of the TikZ diagrams would require frequent requests to the back-end, which increases the required back-end processing capabilities and infrastructure costs. There are possible methods to reduce this effect, but each has additional disadvantages. One option is to provide a button for the user to manually request previews. This would reduce the required back-end processing, but would be less convenient for users than automatic updates to the preview when any changes are made to the TikZ code. Another option is to use a JavaScript graphics package to generate automatic previews in the front-end. This would remove the need for requests to the back-end, but it would result in previews which may be slightly different from the final animation produced by Manim and would be more difficult to implement as it requires parsing functionality in the front-end. While responses to other questions supported the choice of Manim for animation, this feature would be easier to implement when using a JavaScript package (e.g. GSAP) for animation in the front-end.

The possible improvements suggested in the usability study responses are summarised in table 4.1 in order to guide future development of this application. They are compared based on the impact they would have on users and the difficulty of implementation. The impact is determined using both the results of the usability study and subjective estimates. The difficulty is based on research into possible methods of implementation and personal experience in the development of the application.

		Difficulty		
		Easy	Medium	Hard
Impact	High	• Manim configuration options • Variable animation time • Optional line IDs	• `tikz-cd` input • Display helpful error messages • Manually request Manim diagram previews	• Image input • Control node/line movement
	Medium	• Output GIF format • Output series of frames • Ability to remove TikZ diagram input boxes	• More available shapes • Tab for each TikZ diagram on GUI	• Equation input • Output HTML and JavaScript • Automatic previews using JavaScript package
	Low	• Labels on lines • Simple HTML output for embedding mp4 file • Ability to easily load example TikZ • Ability to hide example TikZ		

Table 4.1: Impact and difficulty of possible improvements

Chapter 5

Conclusions

5.1 Decisions and implementation

This project aimed to build a tool [Weetman, 2024] to help users quickly create animations of string diagrams. It was designed for people who have a series of string diagrams in TikZ format and would like to quickly animate changes between the diagrams to be clearer for readers. The tool focuses on being as convenient and automatic as possible rather than providing the user with detailed controls. The usability study results showed that the objectives of this project were met as most features suggested by potential users before seeing the tool were implemented. However, they also showed that additional features for control and configuration would be useful in future versions of the tool.

Multiple possible input formats were compared and TikZ was chosen as the most suitable primarily because of its popularity. The subset of TikZ used by the TikZiT application was focused on as TikZiT is a popular tool which uses a simple, restricted subset of TikZ. The TikZ accepted by the application was extended to also allow FreeTikz syntax, as FreeTikz is another convenient tool designed for string diagrams which uses a restricted subset of TikZ. The results of the usability study strongly supported the decision to accept TikZ and the TikZiT subset as input.

Several methods of making the tool available were compared and online web-based access was chosen as the most suitable option. The reasons for this decision aligned with this tool's aim of user convenience: online web-based access is quick and convenient for users and avoids problems with installation. The results of the usability study strongly supported this decision and the resulting convenience and portability of the application.

The final major design decision was the method of animation. Manim was chosen because it is flexible, free, easy to integrate into a project, and produces a video file of the animation. The usability study results emphasised some of the advantages of Manim: the flexibility, control, and various output options. However, the results also highlighted Manim's disadvantages: it is unable to provide HTML and JavaScript as output and implementing live previews of TikZ diagrams would be particularly difficult.

Many challenges were encountered in the development of this project, some of which are summarised below.

- Lines needed to be converted from TikZ representations using in/out angles and looseness to four points which would be used as handle points of a Bézier curve by Manim.
- FreeTikz compass directions needed to be converted into a point on the edge of a node using a single method for all shapes, including asymmetric shapes.
- In the animation section, one major challenge was the automatic matching of lines in one diagram to lines in the next diagram.
- One challenging conceptual problem was learning React and Django, which required a significant investment of time and effort.
- Finally, the last major challenge was the deployment of the back-end, which was particularly difficult because of Manim's numerous dependencies.

5.2 Further work

Allowing further work has been considered throughout the design and implementation of this project. First, the parsing, Tikz-to-Manim value conversion, and animation were designed to be as modular as possible to easily allow a new parser or animation package to be added. Second, React was chosen as the front-end framework for this application and AWS Elastic Beanstalk was chosen for the deployment of the application back-end partly because their popularity increases the probability that future developers already have experience using them.

Some additional features (e.g. animation preview and subtitles) have already been implemented, but more features are required to ensure the application is as useful as possible for potential users. A range of possible improvements are shown in table 4.1, where the improvements which are closest to the top-left corner should be implemented first as these are low-difficulty high-impact features. The first improvements to implement are adding options for users to specify the length of time spent on each diagram or transition, adding options to configure various aspects of Manim (e.g. background colour), and allowing users to optionally provide lines with ID values.

The next features to implement are the easy-to-implement medium-impact features and the medium-difficulty high-impact features. The easy-to-implement medium-impact features are to allow users to save the output in GIF format or as a series of frames and to allow users to remove TikZ diagram input boxes on the user interface. The medium-difficulty high-impact features are to allow `tikz-cd` input, to display helpful error messages, and to provide users with a button to preview a TikZ diagram as it will appear in the final animation.

Bibliography

Lindsay Bassett. *Introduction to JavaScript object notation: a to-the-point guide to JSON*. O'Reilly Media, Inc, 2015.

David Beazley. Ply. https://www.dabeaz.com/ply/ply.html, 2020.

Sandra Berney and Mireille Bétrancourt. Does animation enhance learning? a meta-analysis. *Computers & Education*, 101:150–167, 2016. ISSN 0360-1315. doi: https://doi.org/10.1016/j.compedu.2016.06.005. URL https://www.sciencedirect.com/science/article/pii/S0360131516301336.

Mike Bostock and Observable, Inc. D3. https://d3js.org/, 2023.

John C. Bowman. Asymptote: Interactive tex-aware 3d vector graphics. *TUGboat*, 31 (2):203–205, 2010.

John C. Bowman and Andy Hammerlindl. Asymptote: A vector graphics language. https://www.math.ualberta.ca/~bowman/publications/asyTUG.pdf, 2008.

Canviz contributors. Canviz. https://code.google.com/archive/p/canviz/.

Bob Coecke, Mehrnoosh Sadrzadeh, and Stephen Clark. Mathematical foundations for a compositional distributional model of meaning, 2010.

Luis Pedro Coelho. Mahotas: Open source software for scriptable computer vision. *Journal of Open Research Software*, Jul 2013. doi: 10.5334/jors.ac.

d3-graphviz contributors. d3-graphviz. https://github.com/magjac/d3-graphviz.

Giovanni de Felice, Alexis Toumi, and Bob Coecke. DisCoPy: Monoidal categories in python. *Electronic Proceedings in Theoretical Computer Science*, 333:183–197, feb 2021. doi: 10.4204/eptcs.333.13. URL https://doi.org/10.4204%2Feptcs.333.13.

Django Software Foundation. Django. https://www.djangoproject.com/, 2024. Version 5.0.3.

Jérome Eertmans. Manim slides: A python package for presenting manim content anywhere. *Journal of Open Source Education*, 2023.

Kjell Magne Fauske. dot2tex - a graphviz to latex converter. https://dot2tex.readthedocs.io/en/latest/, 2006.

Forest contributors. Forest. https://www.ctan.org/pkg/forest, 2017.

Emden R. Gansner, Eleftherios Koutsofios, and Stephen North. Drawing graphs with dot, 2015. URL \url{https://graphviz.org/pdf/dotguide.pdf}.

Pablo Garaizar, Miguel A Vadillo, and Diego López-de Ipiña. Presentation accuracy of the web revisited: animation methods in the html5 era. *PloS one*, 9(10), 2014. ISSN 1932-6203.

Google. Angular. https://angular.io/, 2023. Version 17.0.0.

Alexander Grahn. The animate package, 2023. URL \url{https://tug.ctan.org/macros/latex/contrib/animate/animate.pdf}. Accessed on March 25, 2024.

Inc. GreenSock. Gsap. https://gsap.com/, 2023.

Vincent Göring. Designing a library to create animated sequences using d3.js. Master's thesis, Otto von Guericke University Magdeburg, March 2023.

C. Heunen. Freetikz. https://homepages.inf.ed.ac.uk/cheunen/freetikz/, 2018.

Maxim Houwink. Designing a framework for expressing function block diagrams using json, 2022.

Wei-Chia Huang, Tzung-Sheng Yang, and I-Wei Lai. Efficient explanatory math animation generating with the integration of the step-by-step solver. In *2023 International Conference on Consumer Electronics - Taiwan (ICCE-Taiwan)*, pages 785–786, 2023. doi: 10.1109/ICCE-Taiwan58799.2023.10226735.

ImageMagick Studio LLC. Imagemagick. URL https://imagemagick.org.

Wallace Jackson. *An Introduction to JSON: Concepts and Terminology*, pages 15–20. Apress, Berkeley, CA, 2016. ISBN 978-1-4842-1863-1. doi: 10.1007/978-1-4842-1863-1_2. URL https://doi.org/10.1007/978-1-4842-1863-1_2.

Daniel Jeffries, Raghuveer Mohan, and Cindy Norris. Dsdraw: Programmable animations and animated programs. In *Proceedings of the 2020 ACM Southeast Conference*, ACM SE '20, page 39–46, New York, NY, USA, 2020. Association for Computing Machinery. ISBN 9781450371056. doi: 10.1145/3374135.3385292. URL https://doi.org/10.1145/3374135.3385292.

Stephen C. Johnson. Yacc. https://archive.ph/dfwE9, 2013.

A. Joyal and R. Street. The geometry of tensor calculus, i. *Advances in Mathematics 88.1*, pages 55–112, 1991.

Marin Kaluza, Kresimir Troskot, and Bernard Vukelic. Comparison of front-end frameworks for web applications development. *ZBORNIK VELEUCILISTA U RIJECI-JOURNAL OF THE POLYTECHNICS OF RIJEKA*, 6(1):261–282, 2018. ISSN 1848-1299.

A. Kissinger, A. Merry, C. Heunen, and K. Johan Paulsson. Tikzit. https://tikzit.github.io/, 2020. Version 2.1.6.

Aleks Kissinger. Chyp: Composing hypergraphs, proving theorems. https://act2023.github.io/papers/paper25.pdf, May 2023.

Aleks Kissinger and Vladimir Zamdzhiev. Quantomatic: A proof assistant for diagrammatic reasoning. In Amy P. Felty and Aart Middeldorp, editors, *Automated Deduction - CADE-25*, pages 326–336, Cham, 2015. Springer International Publishing. ISBN 978-3-319-21401-6.

M. E. Lesk and E. Schmidt. Lex. https://archive.ph/aEDH, 2012.

Zhuozhuo Joy Liu and Timothy Sun. Aanim: An animation engine for visualizing algorithms and data structures for educators. In *Proceedings of the 54th ACM Technical Symposium on Computer Science Education V. 2*, SIGCSE 2023, page 1287, New York, NY, USA, 2023. Association for Computing Machinery. ISBN 9781450394338. doi: 10.1145/3545947.3576233. URL https://doi.org/10.1145/3545947.3576233.

Celia Rubio Madrigal. Rendering string diagrams with haskell's diagrams library. Master's thesis, University of Strathclyde, Glasgow, UK, August 2023.

Manim contributors. Manim. https://github.com/ManimCommunity/manim/, 2023.

Richard E. Mayer. Animation as an aid to multimedia learning. *Educational Psychology Review*, 14:87–99, 2002.

Meta. React. https://reactjs.org/, 2022. Version 18.2.0.

Florencio Neves. The tikz-cd package, 2011. URL \url{http://kebo.pens.ac.id/CTAN/graphics/pgf/contrib/tikz-cd/tikz-cd-doc.pdf}. Accessed on March 23, 2024.

nLab authors. homotopy.io. https://ncatlab.org/nlab/show/homotopy.io, October 2023.

Ovidiu Constantin Novac, Damaris Emilia Madar, Cornelia Mihaela Novac, Gyöngyi Bujdosó, Mihai Oproescu, and Teofil Gal. Comparative study of some applications made in the angular and vue.js frameworks. In *2021 16th International Conference on Engineering of Modern Electric Systems (EMES)*, pages 1–4, 2021. doi: 10.1109/EMES52337.2021.9484150.

Pallets. Flask. https://flask.palletsprojects.com/en/3.0.x/, 2024. Version 3.0.2.

Evan Patterson. Knowledge representation in bicategories of relations, 2017.

R. Penrose. Applications of negative dimensional tensors. *Combinatorial Mathematics and its Applications*, pages 221–244, 1971.

Robin Piedeleu and Fabio Zanasi. A Finite Axiomatisation of Finite-State Automata Using String Diagrams. *Logical Methods in Computer Science*, Volume

19, Issue 1, February 2023. doi: 10.46298/lmcs-19(1:13)2023. URL http://lmcs.episciences.org/10963.

Kari Pulli, Anatoly Baksheev, Kirill Kornyakov, and Victor Eruhimov. Realtime computer vision with opencv: Mobile computer-vision technology will soon become as ubiquitous as touch interfaces. *Queue*, 10(4):40–56, apr 2012. ISSN 1542-7730. doi: 10.1145/2181796.2206309. URL https://doi.org/10.1145/2181796.2206309.

Mitchell Riley. Categories of optics, 2018.

P. Selinger. *A Survey of Graphical Languages for Monoidal Categories*, pages 289–355. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010. ISBN 978-3-642-12821-9. doi: 10.1007/978-3-642-12821-9_4. URL https://doi.org/10.1007/978-3-642-12821-9_4.

T. Tantau. Tikz & pgf. https://texample.net/media/pgf/builds/pgfmanualCVS2012-11-04.pdf, 2007.

Till Tantau. Graph drawing in tikz. In Walter Didimo and Maurizio Patrignani, editors, *Graph Drawing*, pages 517–528, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg. ISBN 978-3-642-36763-2.

varkor. quiver: a modern commutative diagram editor, 2020a. URL https://q.uiver.app/.

varkor. Announcing quiver: a new commutative diagram editor for the web, 2020b. URL https://varkor.github.io/blog/2020/11/25/announcing-quiver.html.

Vis.js contributors. Vis.js. https://visjs.org/.

Iain Weetman. Animating string diagrams. https://animate-xsh5.onrender.com, 2024. Code repository at https://github.com/IainWt/animating-string-diagrams/tree/main.

T. Willwacher. Tikzedt. https://code.google.com/archive/p/tikzedt/, 2021.

Tzung-Sheng Yang, Wei-Chia Huang, and I-Wei Lai. Autogeneration of explanatory math animation. In *2022 IEEE International Conference on Teaching, Assessment and Learning for Engineering (TALE)*, pages 727–728, 2022. doi: 10.1109/TALE54877.2022.00128.

Evan You. Vue.js. https://vuejs.org/, 2024. Version 3.4.21.

Mohd Kamir Yusof and Mustafa Man. Efficiency of json for data retrieval in big data. *Indonesian Journal of Electrical Engineering and Computer Science*, 7(1):250–262, 2017.

Appendix A

Participants' information sheet

Project title:	Animating Diagrams
Principal investigator:	Chris Heunen
Researcher collecting data:	Iain Weetman

This study was certified according to the Informatics Research Ethics Process, reference number 837563. Please take time to read the following information carefully. You should keep this page for your records.

Who are the researchers?

Iain Weetman (Cognitive Science BSc student)

What is the purpose of the study?

To assess the usability of the application and provide ideas for improvements or additional features.

Why have I been asked to take part?

You may be able to act as a potential user and provide useful information for the improvement of the application.

Do I have to take part?

No – participation in this study is entirely up to you. You can withdraw from the study at any time, up until June 2024 without giving a reason. After this point, personal data will be deleted and anonymised data will be combined such that it is impossible to remove individual information from the analysis. Your rights will not be affected. If you wish to withdraw, contact the PI. We will keep copies of your original consent, and of your withdrawal request.

What will happen if I decide to take part?

There will be a single session where you attempt to use the application and details related to usability will be recorded. You will also be asked about the possible improvements

to the application.

Are there any risks associated with taking part?

There are no significant risks associated with participation.

Are there any benefits associated with taking part?

No.

What will happen to the results of this study?

The results of this study may be summarised in published articles, reports and presentations. Quotes or key findings will be anonymized: We will remove any information that could, in our assessment, allow anyone to identify you. Your data may be archived for a maximum of 1 years . All potentially identifiable data will be deleted within this timeframe if it has not already been deleted as part of anonymization.

Data protection and confidentiality.

Your data will be processed in accordance with Data Protection Law. All information collected about you will be kept strictly confidential. Your data will be referred to by a unique participant number rather than by name. Your data will only be viewed by the researcher/research team (Iain Weetman, Chris Heunen).

All electronic data will be stored on a password-protected encrypted computer, on the School of Informatics' secure file servers, or on the University's secure encrypted cloud storage services (DataShare, ownCloud, or Sharepoint) and all paper records will be stored in a locked filing cabinet in the PI's office. Your consent information will be kept separately from your responses in order to minimise risk.

What are my data protection rights?

The University of Edinburgh is a Data Controller for the information you provide. You have the right to access information held about you. Your right of access can be exercised in accordance Data Protection Law. You also have other rights including rights of correction, erasure and objection. For more details, including the right to lodge a complaint with the Information Commissioner's Office, please visit www.ico.org.uk. Questions, comments and requests about your personal data can also be sent to the University Data Protection Officer at dpo@ed.ac.uk.

Who can I contact?

If you have any further questions about the study, please contact the lead researcher, Iain Weetman (s1950874@ed.ac.uk).

If you wish to make a complaint about the study, please contact inf-ethics@inf.ed.ac.uk. When you contact us, please provide the study title and detail the nature of your complaint.

Updated information.

If the research project changes in any way, an updated Participant Information Sheet will be made available on `http://web.inf.ed.ac.uk/infweb/research/study-updates`.

Alternative formats.

To request this document in an alternative format, such as large print or on coloured paper, please contact Iain Weetman (s1950874@ed.ac.uk).

General information.

For general information about how we use your data, go to: edin.ac/privacy-research

Appendix B

Participants' consent form

Project title:	Animating Diagrams
Principal investigator (PI):	Chris Heunen
Researcher:	Iain Weetman
PI contact details:	chris.heunen@ed.ac.uk

By participating in the study you agree that:

- I have read and understood the Participant Information Sheet for the above study, that I have had the opportunity to ask questions, and that any questions I had were answered to my satisfaction.
- My participation is voluntary, and that I can withdraw at any time without giving a reason. Withdrawing will not affect any of my rights.
- I consent to my anonymised data being used in academic publications and presentations.
- I understand that my anonymised data will be stored for the duration outlined in the Participant Information Sheet.

Please tick yes or no for each of these statements.

1.	I agree to take part in this study.		
		Yes	**No**

Name of person giving consent		Date		Signature
Name of person taking consent		Date		Signature