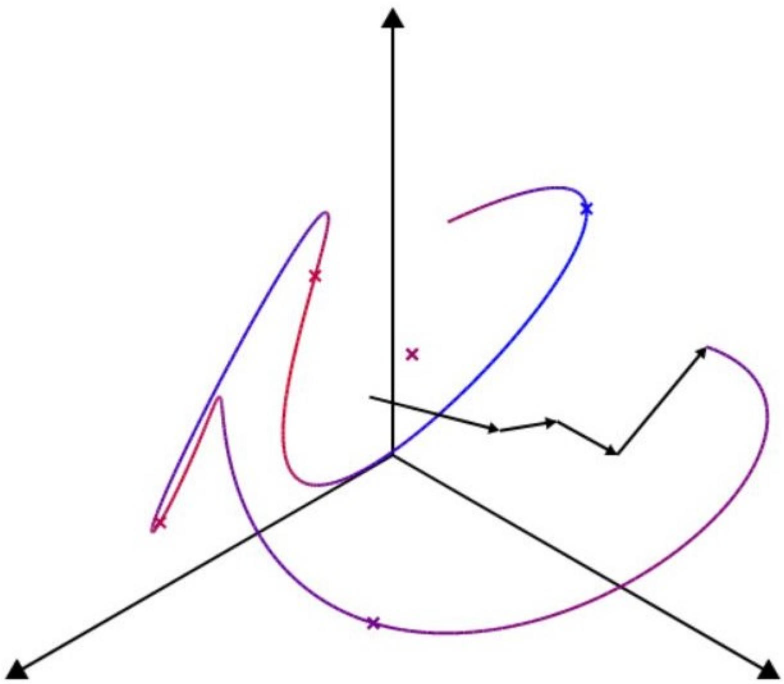
Creating Multidimensional Drawings With Epicycles

Jan Philipp Birmanns, 4a

Supervised by Nicoletta Ravizza-Andri



Kantonsschule Im Lee Winterthur

Matura Project HS2021/22

Winterthur, January 3rd, 2022

Abstract

This paper explores the phenomenon of tracing drawings with epicycles in the two-, three-, and four-dimensional space. The Fourier Transform [1] which is an essential part of today's technology stands at the center of this process. A closer look is taken at both the Discrete Fourier Transform [2] and the Discrete Quaternion Fourier Transform. In order to share the visual intrigue of the transform with readers, two pieces of software have been developed. These can be found at **dft.birmanns.org** and **dqft.birmanns.org**. Through this research, rigorous proofs have been found to explain this behaviour as well as a number of ways to improve the Inverse Discrete Fourier Transform. In order to introduce readers to these findings they will also be familiarized with the underlying mathematical groundwork. This, most importantly, includes complex numbers and quaternions. Thusfar, only few resources exist that discuss epicycles and the Fourier Transform in this context and such detail. This project was inspired by a video created by Grant Sanderson in which he presents epicycles that trace various figures [3].

Preface

At this point I wish to express my appreciation to Nicoletta Ravizza-Andri who not just supervised this project but could aid me through her great interest in mathematics and knowledge of the matter. I am further thankful to Christine Gmür who looked over the sample chapter of this paper. My gratitude is also extended to Emilie Noel Saint Amour and Noah Alexander Birmanns who spent countless hours giving me advice on how to further improve this text. Lastly, I am very grateful for the never-ending support of my parents, especially during the development of this project.

When I first came across Grant Sanderson's video [3] I was immediately intrigued by the complex yet beautiful animations of various epicycles. The mathematics that allow these movements rival if not exceed them in beauty, which thus prompted this research. Many of the theorems and concepts used had previously been unknown to me but soon become rather familiar through the help of such an interesting application. It was further a delight that I could combine my passion for mathematics and computer science through the creation of two pieces of software that allow me to share this phenomenon. This project additionally helped me develop my knowledge of both fields while leading to many joyous moments of discovery.

Contents

List of Figures

1 Introduction

The Fourier Transform is an essential part of modern technology. It is applied to many fields such as communications, astronomy, geology, and optics [4]. Joseph Fourier, a French mathematician and physcist, discovered that any function could be displayed as a combination of sine- and cosine-waves in the early 1800s. This idea would eventually develop into its own field of Fourier-Analysis even though Fourier had initially thought to describe the transfer of heat with it [5]. The transform is so important in today's world, as it allows data signals to be processed and filtered easily.

As this paper will show, a Fourier Transform can also be interpreted as a series of epicycles. This term stems from ancient astronomy and was made famous through Ptolemy's geocentric model. It finds its origin centuries before this system [6], implying that the concept, although very distant from the transform itself, predates most of modern mathematics. Since the discovery of the transform, a range of alternate forms have been developed, such as the Discrete Fourier Transform [2] or the Discrete Quaternion Fourier Transform [7]. These transforms are well-suited for the processing of sets of data as will be done in the following sections.

Alongside this paper two pieces of software have been developed that demonstrate the visual appeal that can attract those unfamiliar with the topic. They can be found on the websites **dft.birmanns.org** and **dqft.birmanns.org**. The first matches the first half of this document where the Discrete Fourier Transform and Inverse Discrete Fourier Transform are discussed. These terms will henceforth be abbreviated as DFT and IDFT respectively. They match the conventional understanding of an epicycle in a two-dimensional space. The second program demonstrates the Discrete Quaternion Fourier Transform and Inverse Discrete Quaternion Fourier Transform which correspond to epicycles in three- and four-dimensional space. These names will be shortened to DQFT and IDQFT throughout this paper. Readers are recommended to experience the programms before moving on to the theory discussed here. Extracts from these programs can also be viewed in sections 8 and 13.

This paper is intended for students that are nearing the end of year twelve and have a general interest in mathematics. For this reason the concept of complex numbers which are vital to this project should be familiar to readers. Nonetheless, important aspects will be redefined as they are utilized throughout the following sections. In order to discuss multi-dimensional drawings which exceed the two-dimensional plane, the quaternion space will also be explored. While a fundamental understanding of quaternions will be of use, it is not necessary to continue reading.

The body of this text can be divided into two similar halves along sections 8 and 9. The first half will start off by defining the term "epicycle" while the second will in turn introduce the quaternion space to the reader. After this the two parts explain how to trace paths in a two- and three-dimensional space accordingly. These sections are followed by proofs and explanations of the corresponding transforms. The former part will additionally discuss methods to improve the Inverse Discrete Fourier Transform. Both halves end by presenting the pieces of software that have been developed to demonstrate the theory.

2 Epicycles

The term “epicycle” does not find its origin in mathematics but stems from astronomy. It was first used by Greek astronomer Apollonius of Perga during the third century BCE [6], making it older than most of modern mathematics. He used the word to describe the motion of a planet that moves on a circle which itself is being carried along the circumference of a larger circle, the deferent [8]. The concept was made world-famous through Ptolemy’s Almagest. At this point it was still believed that the Earth stood still at the center of the universe [6]. Thus, the irregular path taken by bodies such as Mars had been a mystery for decades. Ptolemy found a solution to this problem by proposing that such planets do not move on a regular circle but instead on an epicycle as in figure 1.

While this theory could hold true in the context of a geocentric model, it became obsolete when the heliocentric model was introduced. The true reason for the motion are the varying speeds at which bodies rotate around the sun. For example, whenever Earth passes Mars it seems as if the red planet first changes its direction but then turns around once more to continue its original path. This is only the case from the Earth’s point of view, in actuality Mars simply continues moving on its usual elliptical path [9].

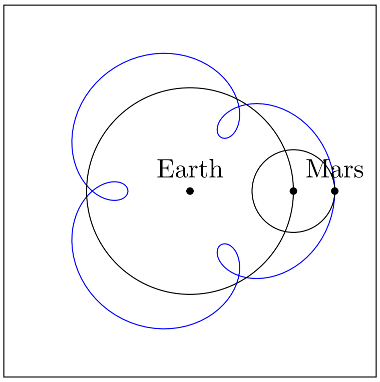


Figure 1: a qualitative representation of the geocentric model

Nonetheless, Ptolemy’s model was highly accurate. The reason for this is that any smooth path can be represented nearly perfectly through epicycles. This was indirectly discovered by Joseph Fourier as a part of Fourier analysis in the early 1800s. He uncovered the so-called “Fourier Transform” which is widely used today. It is based on the idea that any signal can be decomposed into a set of sinosoids and was initially intended to model heat transfer [5]. Today it is most commonly utilized in signal and thus sound processing to decompose signals [4]. The following chapters will step into Ptolemy’s footsteps and make use of the property that epicycles can trace any arbitrary smooth path in the context of the Fourier Transform. They are also often represented through chains of arrows instead of many circles. An individual arrow connects the center of a circle to the next which is moving on its circumference. As the outer circle moves relative to the center of the inner circle, the arrow turns. A more precise approach to this interpretation will be discussed in section 4. Especially in cases where there are many nested epicycles, this method allows a neater visualization.

3 Applying the Fourier Transform to Drawings

One of the prime issues that one faces when attempting to create drawings with the Fourier Transform [1] is that its intended use is to approximate already existing functions. Thus, in order to recreate a drawing with it, a function would first have to be found that connects the infinite amount of points that form such a shape. This, however, is not achievable as the creation of such a function and the gathering of such data would require an unreasonable amount of time. A solution to this problem is the use of approximations. An example would be to represent a drawn line through a sequence of points. These are determined by the position of a pencil or similar at every second during which somebody is drawing this shape. These points are later connected to recreate the original, as shown in figure 2. At a high enough sample rate and slow enough movement the original line can be matched nearly perfectly.

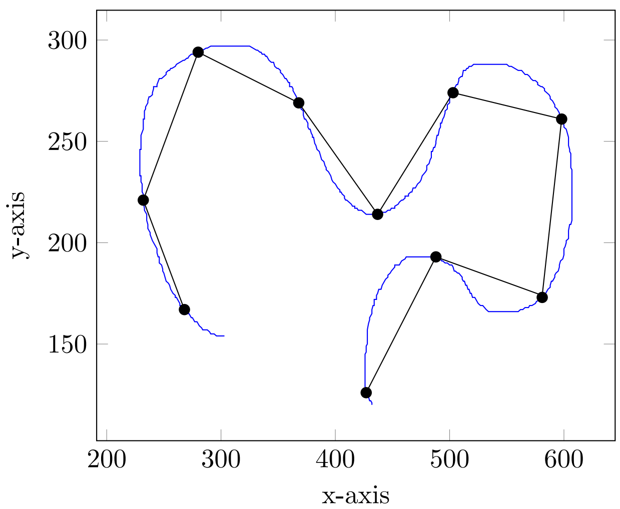


Figure 2: an example of a drawing being approximated by a set of points

Since data points serve as an input rather than mathematical functions, the Fourier Transform no longer applies. Instead, when dealing with individual points, the Discrete Fourier Transform [2] is used:

$$X(k) = \sum_{n=0}^{N-1} x_n \cdot e^{-i2\pi kn\frac{1}{N}}.$$

Just using the DFT in $\mathbb{R}$ will, however, not suffice. Operating in $\mathbb{R}$ allows only one-dimensional input. It is still possible to trace simple drawings or sets of data when the points are ordered so that $n = x$ of a point (x, y) as in figure 3.

pts.	n	x_n
(1,1)	1	1
(2,3)	2	3
(3,2)	3	2
(4,0)	4	0
(5,1)	5	1

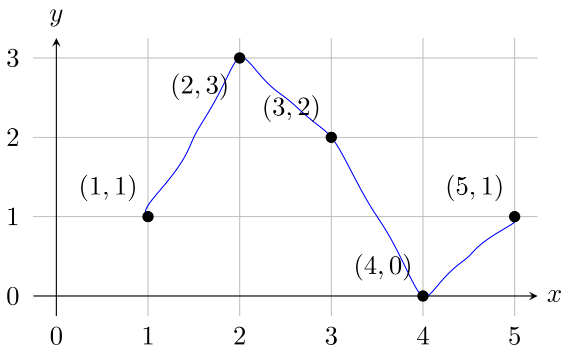


Figure 3 & Table 1: an example set of points being traced by the DFT (& IDFT)

Unfortunately, as soon as the drawn shapes feature two points with the same x-value (such as in loops) several issues come to light. In these cases there are multiple x_n for the same n. Luckily, a very practical trick to work around this problem is to expand the input to two dimensions: the two-dimensional set $\mathbb{C}$. $\mathbb{C}$ describes the set of all complex values which are commonly denoted as "$a + bi$" (in Cartesian form). A projection $\phi : \mathbb{R}^2 \rightarrow \mathbb{C}$ is then defined which converts a point (x, y) to $x + yi$. More complex shapes can then be traced as presented in figure 4:

pts.	n	x_n
(1,4)	1	1+4i
(2,4.5)	2	2+4.5i
(4,4)	3	4+4i
(3.5,2)	4	3.5+2i
(3,1)	5	3+1i
(2,0.5)	6	2+0.5i
(1,2)	7	1+2i
(2,3)	8	2+3i

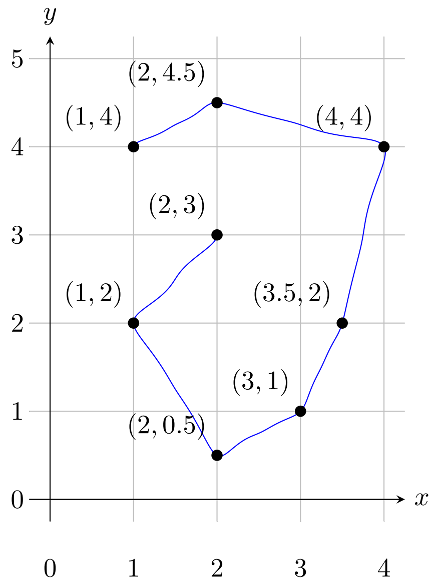


Figure 4 & Table 2: an example set of points being traced by the DFT (& IDFT)

Fortunately enough, the DFT is already capable of handling complex values [1] which means that it can remain unchanged. Methods of handling the output of the DFT to receive this approximation and a proof of the transform that applies to $\mathbb{R}$ and $\mathbb{C}$ will be discussed in section 5.

4 Interpretation of the IDFT as a Set of Arrows

At the heart of the visualization of the Fourier Transform in a two-dimensional space lies the interpretation of the Inverse Discrete Fourier Transform [2] as a set of arrows. This intially unintuitive connection will be discussed in the following section. Commonly, the IDFT is expressed as

$$x(k) = \frac{1}{N} \sum_{n=0}^{N-1} X_n e^{i2\pi nk\frac{1}{N}}.$$

As section 5 will discuss further, X_n describes complex constants which have already been collected, using the Discrete Fourier Transform. These complex values are then multiplied with $e^{i2\pi nkN^{-1}}$ and divided by N to calculate the final point. To make the connection more explicit, the form of the two factors that are being observed, $e^{i2\pi nkN^{-1}}$ and X_n, are altered. While complex values of the traditional form "$a + bi$" are already sufficiently defined, an alternative notation exists. Figure 5 shows the geometrical interpretation of a point of form "$a + bi$". This structure is also referred to as the Cartesian form. As figure 6 shows, a complex value can be defined through its distance and angle to the origin as well. This alternative form is referred to as the Polar form.

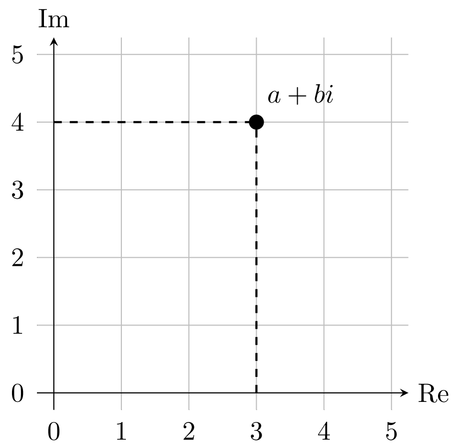


Figure 5: the complex value $3 + 4i$ in the complex plane

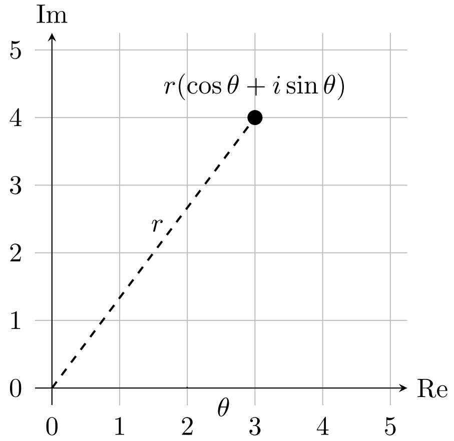


Figure 6: the complex value $3+4i$ in the complex plane

Instead of seeing such values as points that are defined by the distance r and angle θ, they can be understood as the tips of arrows of length r that have been turned by θ.

Similarly, the form of $e^{i2\pi nkN^{-1}}$ can be changed. For this, Euler's formula [10] is applied. The equation states that $e^{ix} = \cos x + i \sin x$ and thus allows the following transformation:

$$e^{i2\pi nkN^{-1}} = \cos(2\pi nkN^{-1}) + i\sin(2\pi nkN^{-1}).$$

Now that the factors have been converted into more suitable forms, they can, once more, be compared. The IDFT equals:

$$x(k) = \frac{1}{N}\sum_{n=0}^{N-1}(r_n(\cos\theta_n + i\sin\theta_n)) \cdot (\cos(2\pi nkN^{-1}) + i\sin(2\pi nkN^{-1})).$$

The two values can be multiplied which each other and return

$$x(k) = \frac{1}{N}\sum_{n=0}^{N-1} r_n((\cos(\theta_n)\cos(\omega) - \sin(\theta_n)\sin(\omega)) + i(\cos(\theta_n)\sin(\omega) + \sin(\theta_n)\cos(\omega)))$$
$$\text{with } \omega = 2\pi nkN^{-1}.$$

Making use of the trigonometric addition formulas [11], this can be simplified to

$$x(k) = \frac{1}{N}\sum_{n=0}^{N-1} r_n(\cos(\theta_n + \omega) + i\sin(\theta_n + \omega)) \quad \text{with } \omega = 2\pi nkN^{-1}.$$

As shown, the value of the multiplication $X_n \cdot e^{i2\pi nkN^{-1}}$ is simply a complex number (in Polar form) which in turn can be understood as an arrow of the length r_n with the angle $\theta_n + 2\pi nkN^{-1}$. Additionally, the fact that it is part of a sum implies that the entire IDFT can be understood as a chain or series of arrows. Each one of them is connected to the previous through its base and the following through its head.

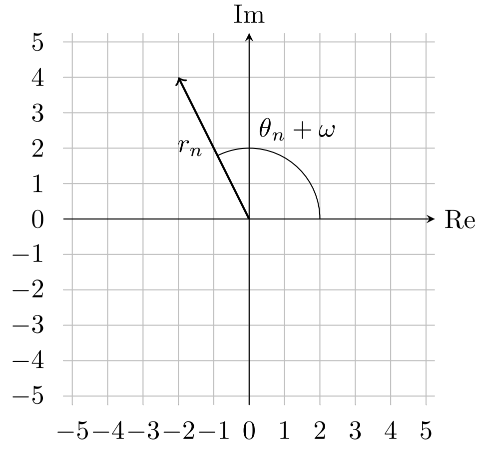


Figure 7: an example of a single arrow determined by a summand of $x(k)$

Furthermore, the values of the angle and length of these arrows must be determined. The length r_n can easily be computed as $r_n = |X_n|$. While the first element of the angle (θ_n) is simply $\arctan(\Im X_n / \Re X_n)$, finding ω becomes more difficult. When k is set so that $k \in \mathbb{N} \cup \{0\}$ and $k \leq N$, it returns the original values $x_0, x_1, x_2, ...$, depending on which k is selected. However, what happens when k is not within those boundaries?

First, the outcome is considered when $k > N$. This would imply that $k \cdot N^{-1} > 1$. An important property of trigonometric functions is that (if $f(x)$ is a trigonometric function) $\exists a \in \mathbb{R} : f(x) = f(x-a)$. Generally, functions with this property are called periodic. When $\omega = 2\pi nkN^{-1}$ is plugged into a single summand of $x(k)$, it thus follows

$$\cos(\theta_n + 2\pi n\frac{k}{N}) + i\sin(\theta_n + 2\pi n\frac{k}{N}) = \cos(\theta_n + 2\pi n\frac{k-N}{N}) + i\sin(\theta_n + 2\pi n\frac{k-N}{N}).$$

This implies that once $k > N$, the IDFT returns to the beginning, creating an endless loop. Due to all trigonometric functions having the range $\mathbb{R}$, it is clear that the IDFT will create a continuous path between every point $x_k : k \in \mathbb{N} \wedge k \leq N$. This means that there are even values x_k $\forall k \in \mathbb{R}$. Yet another important value in ω is n. It determines the frequency at which the arrow spins. There exists one arrow of each whole number frequency between zero and N.

5 The Magic Behind the Discrete Fourier Transform

The Inverse Discrete Fourier Transform [2], or IDFT in short, is the opposite of the DFT and expresses every value $x(k)$ and thus x_n as follows:

$$x(k) = \frac{1}{N}\sum_{n=0}^{N-1} X_n e^{i2\pi nk\frac{1}{N}} = \frac{1}{N}(X_0 e^{i2\pi 0k\frac{1}{N}} + X_1 e^{i2\pi 1k\frac{1}{N}} + \cdots + X_{N-1}e^{i2\pi(N-1)k\frac{1}{N}}). \tag{1}$$

One of the most important properties of the IDFT is that while the DFT has a domain of $k \in \mathbb{N}$, it has the range $\mathbb{R}$. It also true that every set of points x_n can be expressed through the IDFT, given the correct selection of coefficients X_n in the formula. The values $n, k,$ and N are already given by the equation with N equaling the number of data points. This means that the goal of the DFT is to filter out these X_n from a given data set. The following passage will try to demonstrate how the DFT accomplishes this and to ultimately prove the validity of the DFT.

5.1 Proof of the DFT

As shown before, the DFT is equal to

$$\sum_{n=0}^{N-1} x_n \cdot e^{-i2\pi kn\frac{1}{N}}. \tag{2}$$

The equation of the IDFT (equation 1) can be inserted into the DFT (equation 2), as it simply expresses the values x_n in an alternative form:

$$\sum_{n=0}^{N-1}(\frac{1}{N}\sum_{m=0}^{N-1} X_m e^{i2\pi mn\frac{1}{N}}) \cdot e^{-i2\pi kn\frac{1}{N}}$$

$$= \frac{1}{N}\sum_{n=0}^{N-1}(X_0 e^{i2\pi 0n\frac{1}{N}} + X_1 e^{i2\pi 1n\frac{1}{N}} + \cdots + X_k e^{i2\pi kn\frac{1}{N}} + \cdots + X_{N-1}e^{i2\pi(N-1)n\frac{1}{N}}) \cdot e^{-i2\pi kn\frac{1}{N}}.$$

The exponents cancel out for the single summand where $m = k$ which thus equals X_k or $N \cdot X_k$ once the values have been summed up. This still leaves behind a series of

$$X_m e^{i2\pi mn\frac{1}{N}} e^{-i2\pi nk\frac{1}{N}} = X_m e^{i2\pi n\frac{1}{N}(m-k)}$$

where $m \neq k$. These have to amount to zero for the equation to return X_k. To prove that this is in fact true, one must take one more piece of information from the DFT. A few transformations show that

$$\sum_{n=0}^{N-1}(\frac{1}{N}\sum_{m=0}^{N-1} X_m e^{i2\pi n(m-k)\frac{1}{N}}) = \frac{1}{N}\sum_{m=0}^{N-1}(\sum_{n=0}^{N-1} X_m e^{i2\pi n(m-k)\frac{1}{N}}). \tag{3}$$

This implies that one can also view a single $X_m e^{i2\pi n(m-k)\frac{1}{N}}$ as n varies. Geometrically, one such sum expresses movement along a circle of radius $|X_n|$ in steps of $2\pi\frac{m-k}{N}$ [3] which will henceforth be denoted as α. To understand this interpretation, one should be aware of Euler's formula [10] which states $e^{ix} = \cos x + i \sin x$.

As figure 8 demonstrates, the values $X_m e^{n\alpha}$ will add up to zero as n moves from 0 to $N-1$. This demonstrates that $\sum_{n=0}^{N-1} X_m e^{in\alpha} = 0$ for $m \neq k$. In this example $X_m = 3 + 4i$, $(m - k) = 1$, and $N = 8$. The same can be done for k is picked so that $m - k = 0$. Such an example can be viewed in figure 9. As $\alpha = 0$, the different summands will equal the same value X_k for all n and add up to $N \cdot X_k$. Thereby it has been shown that

$$\frac{1}{N}\sum_{m=0}^{N-1}(\sum_{n=0}^{N-1} X_m e^{i2\pi n(m-k)\frac{1}{N}}) = \frac{1}{N}\sum_{m=0}^{N-1} X_k = X_k.$$

which completes the more intuitive approach to proving the DFT.

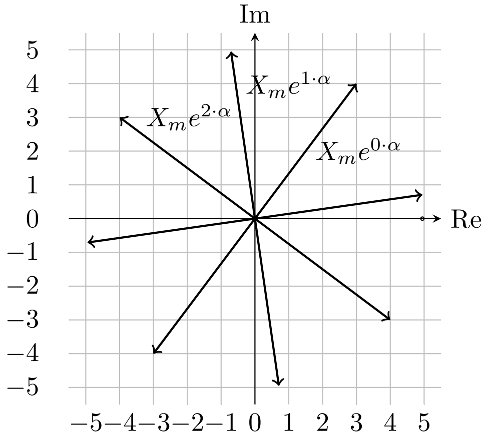


Figure 8: an example for different $X_m e^{in\alpha}$ as n varies and $\alpha = 2\pi\frac{m-k}{N} = 2\pi\frac{1}{8}$

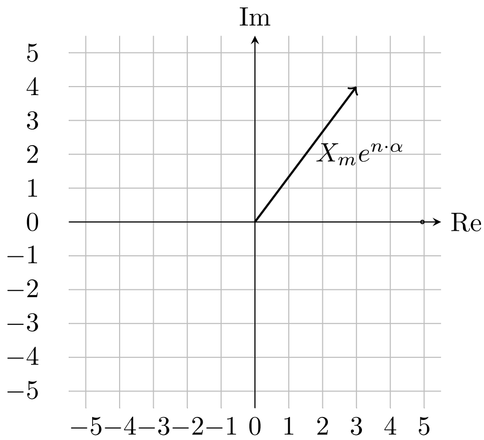


Figure 9: an example for different $X_m e^{in\alpha}$ as n varies and $\alpha = 2\pi\frac{m-k}{N} = 0$

Additionally, there exists a more rigorous proof to achieve this last step. The inner sum of equation 3 is altered in the following way:

$$\sum_{n=0}^{N-1} X_m e^{i2\pi n(m-k)\frac{1}{N}} = X_m e^{-i2\pi nk\frac{1}{N}} \sum_{n=0}^{N-1} e^{i2\pi nm\frac{1}{N}} \stackrel{?}{=} 0.$$

It has to be shown that the product does, in fact, equal zero when $m \neq k$. As three values are being multiplied with each other, at least one of them has to equal zero for this to be true. Since this argument has to be true for any X_m, the coefficent cannot be zero. The second factor, $e^{-i2\pi nk\frac{1}{N}}$, has to be larger than zero because for any value n in $\mathbb{R}$, $e^n > 0$. This leaves the proof of

$$\sum_{n=0}^{N-1} e^{i2\pi nm\frac{1}{N}} \stackrel{?}{=} 0.$$

Since this is a geometric series of form $\sum_{i=0}^{n} a_i r^k$, the geometric sum formula [11] can be applied. It states that for any geometric series [11], its sum equals $a_0 \frac{1-r^n}{1-r}$. Additionally, Euler's formula [10] implies that $e^{i2\pi 0m\frac{1}{N}} = e^{i2\pi Nm\frac{1}{N}}$. This gives

$$\sum_{n=0}^{N-1} e^{i2\pi nm\frac{1}{N}} = \sum_{n=1}^{N} e^{i2\pi(n-1)m\frac{1}{N}} = \sum_{n=1}^{N} 1 \cdot (e^{i2\pi m\frac{1}{N}})^{n-1} = 1 \cdot \frac{1-(e^{i2\pi m\frac{1}{N}})^N}{1-e^{i2\pi m\frac{1}{N}}} = \frac{1-e^{i2\pi m}}{1-e^{i2\pi m\frac{1}{N}}}$$

Euler's formula also shows that $e^{i2\pi m} = \cos(2\pi m) + i\sin(2\pi m) = 1$ if $m \in \mathbb{Z}$. For the previous equation, this implies

$$\sum_{n=0}^{N-1} e^{i2\pi nm\frac{1}{N}} = \frac{1-e^{i2\pi m}}{1-e^{i2\pi m\frac{1}{N}}} = \frac{1-1}{1-e^{i2\pi m\frac{1}{N}}} = 0.$$

This new piece of information completes the last step of this proof. When applied to equation 3, one receives

$$\frac{1}{N}\sum_{m=0}^{N-1}(\sum_{n=0}^{N-1} X_m e^{i2\pi n(m-k)\frac{1}{N}}) = \frac{1}{N}\sum_{m=0}^{N-1}(X_k e^{i2\pi n(k-k)\frac{1}{N}}) = \frac{1}{N}\sum_{m=0}^{N-1}(X_k \cdot 1) = X_k.$$

Thereby, it has been rigorously shown that the Discrete Fourier Transform can filter out X_k for any suitable k from a set of data.

5.2 Example

For the sake of clarity, the Discrete Fourier Transform will be performed on an example set of data. For this four evenly spaced points on an ellipse have been chosen. The exact values are given in table 3 and figure 10. From this set follows that $N = 4$.

pts.	x_n
(4.619,1.148)	$4.619 + 1.148i$
(-1.913,2.772)	$-1.913 + 2.772$i
(-4.619,-1.148)	$-4.619 - 1.148i$
(1.913,-2.772)	$1.913 - 2.772i$

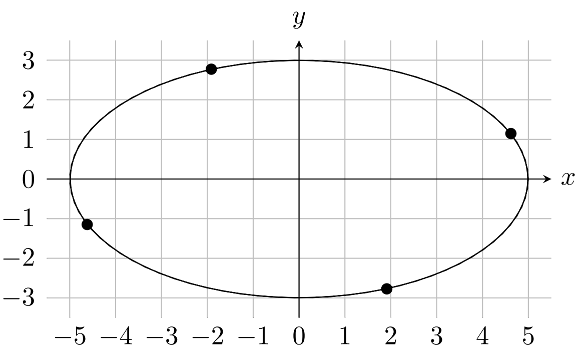


Figure 10 & Table 3: an example set of data

In a first step the coefficient X_0 is calculated. As the DFT states

$$X(0) = \sum_{n=0}^{3} x_n \cdot e^{-i2\pi 0n\frac{1}{4}} = \sum_{n=0}^{3} x_n.$$

For the given values this equals

$$X(0) = (4.619 + 1.148i) + (-1.913 + 2.772i) + (-4.619 - 1.148i) + (1.913 - 2.772i) = 0.$$

Since X_0 is the arrow of frequency zero, it represents the rigid point that the other moving arrows will connect to. This allows the construction to be moved quite easily by just adjusting X_0. It is often not displayed in visualizations of the IDFT as epicycles or a series of arrows since it does not move. The value of X_0 mathematically simply expresses the sum of all points or the average once it has been divided by N in the IDFT. Coefficient X_1 is equal to

$$X(1) = \sum_{n=0}^{3} x_n \cdot e^{-i2\pi 1n\frac{1}{4}} = 3.696 + 1.531i.$$

Similarly, the remaining X_n can be calculated, giving $X_2 = 0$ and $X_3 = 0.924 - 0.383i$. Together the different coefficients give:

$$x(k) = (3.696 + 1.531)\frac{1}{4}e^{2\pi\frac{k}{4}} + (0.924 - 0.383i)\frac{1}{4}e^{2\pi 3\frac{k}{4}}.$$

It can easily be confirmed that this in fact holds true for $x_0, x_1, x_2,$ and x_3. Plotting this equation for $k \in [0; 4]$ reveals that the equation does not trace the ellipse but instead chooses a more unelegant path. The graph can be viewed in figure 11. Methods to visually improve the DFT to accomplish this will be discussed in section 6.

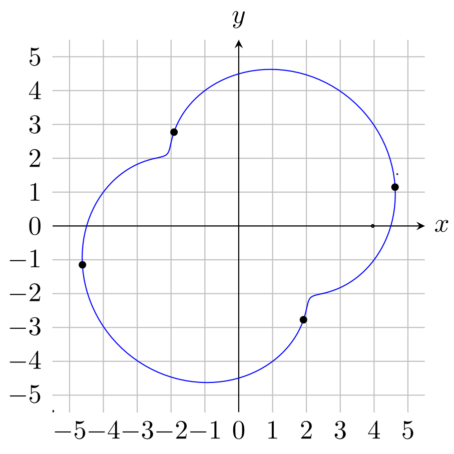


Figure 11: the IDFT of an example set of data

6 Improving the Discrete Fourier Transform

The Discrete Fourier Transform is best suited to process signals [4] and not to draw shapes. Thus there are various improvements that can be made to enhance the visual experience at the cost of precision. When one uses the unchanged DFT and IDFT with an unaltered set of data, drawings become unrecognisable. Such an example can be viewed in figure 12. An IDFT will, in its original form, require a single loop per point, making it unsuited for most drawings. Although it still runs through every point, it is far from accurately resembling the inteded shape. Various changes can be made to improve the final image.

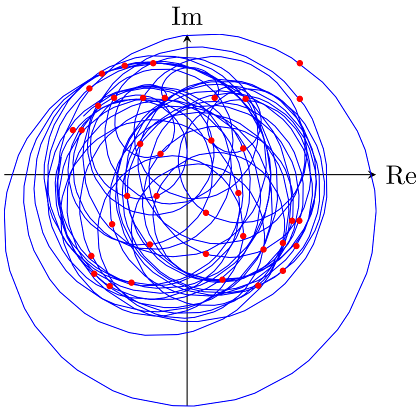


Figure 12: an example of an unchanged IDFT running through a given set of points

6.1 Arrows of Negative Frequencies

Even when viewing the movement of a system of very few epicycles, chaotic activity can arise. They feature many spirals that are created every time an epicycle completes a rotation before its deferent. These are the core issue as they distract from the points that form the original shape. Such issues can be circumvented by pairing up every arrow with another one that turns in the opposite direction [12]. In figure 13 the movement of a single arrow can be compared to the paths of chains of two arrows that add up to the length of the first.

When both arrows are of equal length they end up creating a simple line. This phenomenom can be explained through Euler's formula [10] which implies the following:

$$e^{i2\pi\frac{kn}{N}} + e^{-i2\pi\frac{kn}{N}} = \cos(2\pi\frac{kn}{N}) + i\sin(2\pi\frac{kn}{N}) + \cos(-2\pi\frac{kn}{N}) + i\sin(-2\pi\frac{kn}{N}) = 2\cos(2\pi\frac{kn}{N}).$$

The chain of arrows loses any imaginary component, from which follows that their sum only moves on the real axis. Additionally, it equals the real component of $2e^{i2\pi\frac{kn}{N}}$, describing an arrow that is twice as long as one of the original summands. By multiplying the components with a coefficient X_n, the direction and length of the line can be determined.

When the arrows are of unequal length they create an ellipse. It has a width of $u + v$ and a height of $u - v$ when u is the length of the longer arrow and v the length of the shorter one. Such an observation can also be explained with the help of Euler's formula [10]:

$$\begin{aligned} ue^{i2\pi\frac{kn}{N}} + ve^{-i2\pi\frac{kn}{N}} &= u\cos(2\pi\frac{kn}{N}) + ui\sin(2\pi\frac{kn}{N}) + v\cos(-2\pi\frac{kn}{N}) + vi\sin(-2\pi\frac{kn}{N}) \\ &= (u+v)\cos(2\pi\frac{kn}{N}) + i(u-v)\sin(2\pi\frac{kn}{N}). \end{aligned}$$

It follows that by splitting a single arrow into two with opposite frequencies, the total path can become severely less chaotic.

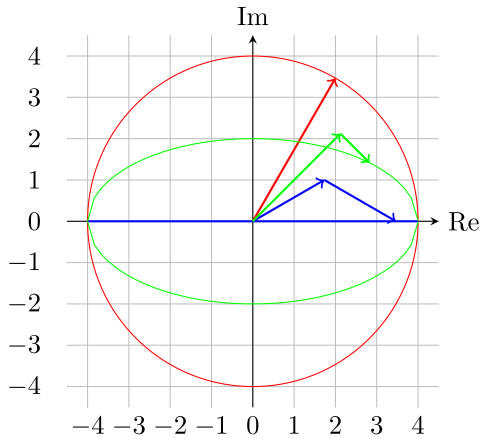


Figure 13: comparing the path of a single arrow to chains of two arrows

This idea can also be applied to the Fourier Transform. The IDFT is then equal to

$$x(k) = \frac{1}{N} \sum_{n=-N+1}^{N-1} X_n e^{i2\pi nk\frac{1}{2N-1}}.$$

Its counterpart, the DFT, sees a change in its domain which is equal to $\{-N+1, -N+2, \cdots, N-1\}$ instead of $\{0, 1, \cdots, N-2, N-1\}$. For every X_n there thus exists a X_{-n} with an according arrow that spins in the opposite direction. It is important to notice that X_n does not equal $-X_{-n}$ since $e^x \neq -e^{-x}$. This strategy improves the result greatly as can be seen in figure 14. Nonetheless, the final image has a rounded shape which can be reduced through another trick.

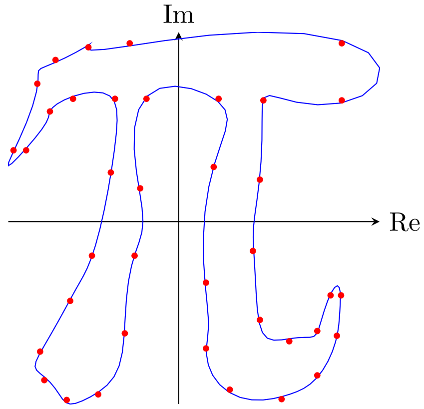


Figure 14: an example of the IDFT improved through arrows of negative frequencies

6.2 Generating Additional Data

Since the Fourier Transform is being used to recreate drawings in this case, visual appeal as opposed to accuracy becomes the main focus. This allows the generation of additional data that will improve the look of the result. Currently, the path taken by the chain of arrows in between the individual points is completely free and thus often curves instead of remaining straight. Additional coordinates located on the line between two points of the given data can be added, restricting the motion of the arrows to more closely follow this line. The simplest method is adding the middlepoints of each pair of adjacent points to the dataset. Expressed mathematically, with A being the original set of points and A' the altered, this is

$$A' = A \cup \{x = (x_n + x_{n+1})/2 : x_n, x_{n+1} \in A\} \cup \{(x_{N-1} + x_0)/2\}.$$

This process can be repeated which will lead to further straigtening of the connections. As figure 15 shows, it can result in a near perfect representation of a given shape even after only two cycles of generating additional data. One downside is that points of organic shapes and poorly sampled curves will, of course, also be connected through straight lines even though the intended drawing may have been different. However, this strategy does prove particularly useful for poorly sampled presets (such as the pi example in figure 15) as they often contain little information and many straight lines.

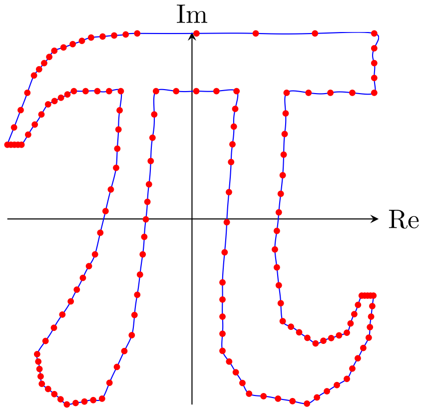


Figure 15: an example of the IDFT improved through generated points

6.3 Variable Precision

Mainly focusing on visual appeal instead of precision brings further options to light. While the Fourier Transform can exactly trace a determined set of points, there are cases in which such precision is not needed. Conventionally, the values X_k are calculated for all $k \in \mathbb{Z} : |k| < N$. The more X_k that are used in the final IDFT, the more accurate it becomes. Thus, it is possible to use less at the cost of precision. As presented in figure 16, this cost is very small. Even when using just 50 out of 152 coefficients, which is represented through the red line, only minor differences can be detected. These become almost inexistant once 100 of 152 are present (blue). Mathematically, this change restricts the domain of the DFT and alters the IDFT to the following

$$x(k) = \frac{1}{N} \sum_{n=0}^{m-1} X_n e^{i2\pi nk \frac{1}{2N-1}}.$$

where m is the number of coefficients.

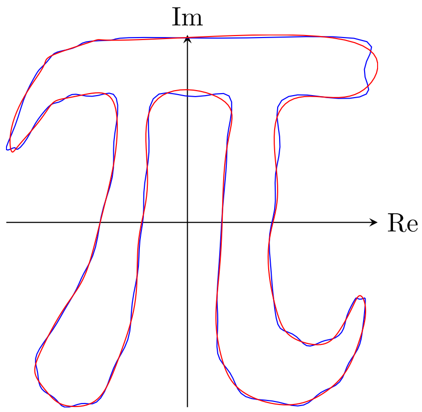


Figure 16: an example of IDFTs of varying accuracy

6.4 Sorting

A further visual enhancement that can be made is sorting arrows by size. This corresponds to ordering the coefficients X_n by magnitude $|X_n|$. Such methods do not effect the final path. However, they have the advantage that by moving larger values to the front, most of the displacement is completed after the first few arrows. Due to their length the majority of movement can be observed much more easily. Shorter arrows will in turn collect at the end of the chain.

The values X_n have the advantage that with increasing n their magnitude decreases. This follows from the fact that to find X_n every value x_n is multiplied by $e^{-i2\pi\frac{nk}{N}}$ which is inversely proportional to n. However, this does not imply that $X_n > X_{n+1}$ for every suitable n. The change corresponds to a downwards trend rather than a strict order. By sorting the arrows, slight outliers can be put back into place. The coefficient X_0 is excluded from these processes. There is a wide range of sorting algorithms that could be applied to this case. Some examples for simple solutions are: Bucket Sort, Bubble Sort, and Counting Sort [13]. It is important to keep in mind that once the order has been changed, implying that $X_n \neq X'_n$ for at least one n, the IDFT must be adjusted such that the frequency still matches with the correct coefficient.

7 Automization of the DFT and IDFT

This project is accompanied by two pieces of software that demonstrate the theory of Fourier Transforms. The first presents the aforementioned Discrete Fourier Transform. It allows a user to create a drawing of their pleasing or pick from a range of examples which will then be traced by an epicycle. JavaScript was chosen as the programming language, CSS and HTML were used to describe the user interface. The entire code can be found in appendix A. In order to make the program as accessable as possible, it has also been uploaded to **dft.birmanns.org**. Demonstrations can be found in section 8.

7.1 Usage

The user is presented with two options of input. The first option is to select one of the two given examples. The first provides a pi-symbol, the second a logo previously used by the Kantonsschule Im Lee which features the main building of the school. Both are loaded from txt-files that store the coordinates that make up these shapes. This allows for simple modification and future addition of further examples. Alternatively, the user can create a drawing themselves, using a mouse or touchscreen. Every time the pen moves, a new data point is added to an array. It can be reset with the press of an additional button located to the right of the "Run Calculation" button. Both examples and a drawing can be viewed in 8.

In the second step they can pick the amount of arrows that the final epicycle will consist of. This value corresponds to the total number of coefficients X_n. Given that the DFT can only find values up to X_N, the user is limited by the length of the data set. As they alter their decision, the software shows the arrows in their initial position along with the exact points that have been selected in the previous step. The chain of arrows is created through a custom class that simply requires the coefficients calculated through the DFT.

Once the confirm button has been pressed, the program moves to the final presentation of the IDFT. As the arrows move, the last one is followed by a trail that runs through the previous points. Additionally, the original drawing is shown, allowing a direct comparison. The movement can be stopped with the pause button located at the bottom of the screen. Using the one next to it, the user can reset the program and repeat the process with a different set of data.

7.2 Alterations

While the software is not demanding in any way to most computers, the IDFT can be altered in code to be understood more easily. As section 4 has shown, it can be interpreted as a set of arrows. This idea can be translated to JavaScript. When an instance of the class that constructs the arrows is built, an object is generated with it. Upon its creation, the DFT is called to calculate the different X_n. These are then used to find the initial angle and length of the arrows that will make up the epicycle. Together with the matching frequency, these numbers are stored in the new object. A further property is added to track the position that is being pointed at.

Whenever the screen refreshes the different angles are altered by a fixed amount multiplied by their frequency. The position that the arrow points at is changed accordingly. This value is irrelevant to the arrow itself as it is sufficiently defined by length and angle, however, it is useful to the next one. The following arrow can use it to determine its global position. Its base matches the location of the previous arrow's head or where it is pointing to. Using length and angle the relative position of the head to the base can then be calculated. This process makes especially the tracking of the path of an individual arrow much less tedious and more efficient. Otherwise this would have to be done by calculating and subtracting two seperate IDFTs.

Another advantage of this method is that it allows the implementation of various improvements proposed in section 6. The arrows can be assigned a precise order within the object that stores the various values. Since the lengths have already been determined in a prevous step this process is equivalent to using a sorting a algorithm that arranges the arrows according to these values. In this case the Bubble Sort algorithm has been chosen. It repeatedly passes over the sequence and compares two neighboring values with each other in each step. If they are in the incorrect order their positions are swapped [13]. While the operation only has an efficiency of $O(n^2)$, it suffices for this application. Once this step has been completed and each arrow has an according index, the user can decide how many of these are utilized. The selection is limited to even numbers as for every arrow that turns clockwise their must be one that turns in the other direction.

7.3 Further Development

Throughout the time during which this project was created, a program could be developed that successfully demonstrates the beauty that lies within the Fourier Transform. Nonetheless, there are various features that could not be completed within the given time frame. Some of the lacking conveniences are further examples or various toggles to customize the final view. The most apparent issue is the support for sketches that consist of multiple non-continuous lines. Even though the program will still return a valid result, these more often than not will consist of much erratic behaviour. This stems from the fact that the points will be traced in the order they were drawn. In most cases this is far from optimal and can create unwanted lines. A solution to this is to, before processing, find the shortest path that runs through all values. Unfortunately, this category of problem takes up a large section of mathematics and could thus not be covered as a part of this project.

8 Examples in Two-Dimensional Space

This section presents screenshots of the software described in section 7. QR codes are located underneath each image that will lead to videos of the respective epicycles in motion. The first demonstration shows the creation of a custom drawing that is then traced through an epicycle consisting of 201 arrows. The video features the entire process of creation, customization, and viewing.

Figure 17: a screenshot of an epicycle tracing a drawing

Figure 18: https://youtu.be/RZB9pb-wBVs

The second features the greek letter pi. This epicycle is also one of the examples that can be selected in the program. It is made of 152 arrows.

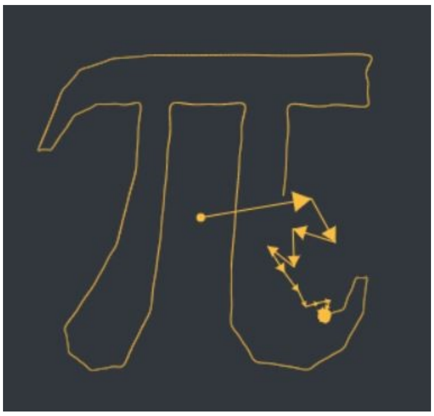


Figure 19: a screenshot of an epicycle tracing pi

Figure 20: https://youtu.be/1d6mCSeMxlk

In the last sample the former logo of the Kantonsschule Im Lee can be seen. It, as well, is one of the examples found in the software. The epicycle is composed of 96 arrows.

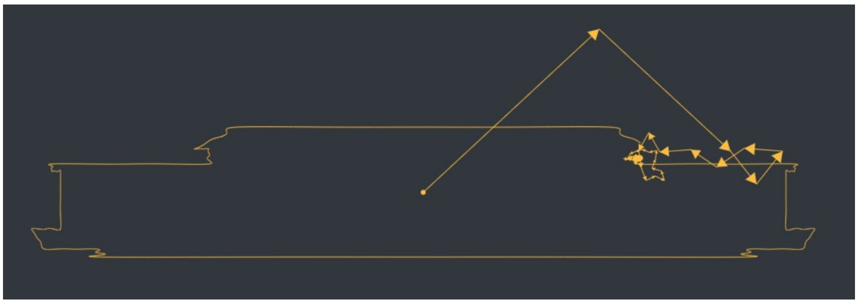


Figure 21: a screenshot of an epicycle tracing a logo

Figure 22: https://youtu.be/lSeHVt1KCTQ

9 Introduction to Quaternions

The following sections will make use of quaternions which will thus be introduced here.

9.1 Concept

Similarly to complex numbers being an expansion of real numbers, quaternion numbers form a further expansion of the complex numbers into a four-dimensional space $\mathbb{H}$. They find a wide range of applications in modern technology where they are most often used to calculate rotations in three-dimensional space. The values are then referred to as Euler Angles [14]. Sir William Rowan Hamilton was an Irish mathematician that developed the system of quaternions in 1843. He had sought to find a method of describing three-dimensional problems in mechanics. After years of struggle he found that by adding a fourth dimension, the normal laws of algebra could be maintained except for communativity [15]. Instead of just using the imaginary number $i = \sqrt{-1}$, these numbers are made up of two further imaginary dimensions: j and k. A quaternion q has the structure

$$q = a + bi + cj + dk.$$

In this representation $a, b, c,$ and d are real numbers, $i, j,$ and k are referred to as basic quaternions. It is made up of a scalar part a and a vector part $bi + cj + dk$. These terms are often shortened as $\mathrm{Sc}(q)$ or q_0 and $\mathrm{Vec}(q)$ respectively [16]. While simple addition and subtraction remain unchanged with

$$q_1 + q_2 = (a_1 + a_2) + (b_1 + b_2)i + (c_1 + c_2)j + (d_1 + d_2)k,$$

multiplication and division are altered. Multiplication in quaternion space is defined in the following way [16]:

$$ij = k, ji = -k, \quad jk = i, kj = -i, \quad ki = j, ik = -j.$$

Most importantly [16],

$$i^2 = j^2 = k^2 = ijk = -1.$$

As stated before, the quaternion space is thus non-communative. As in $\mathbb{C}$, conjugates play an important role. The conjugate of a quaternion q is

$$\bar{q} = a - bi - ci - di$$

and is often represented through $\bar{q}$ [16]. The norm on the other hand is simply

$$|q| = \sqrt{q\bar{q}} = \sqrt{a^2 + b^2 + c^2 + d^2}.$$

Since the quaternion space is made up of four dimensions, it can also be interpreted as a three dimensional geometric space as Sir Hamilaton initially intended. This implies that a single quaternion can be used to represent a point in space that would usually require three values $x, y,$ and z. Such interpretations will be used in the following sections, usually the real dimension is excluded. An example can be seen in figure 23.

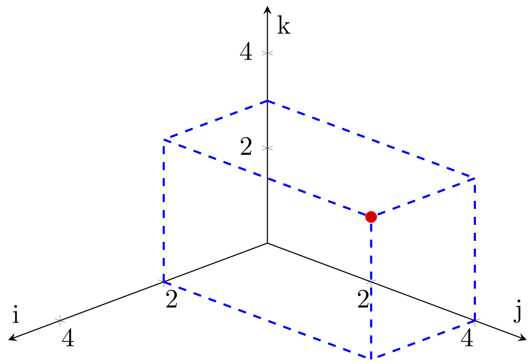


Figure 23: a possible interpretation of $0 + 2i + 4j + 3k$ in space

9.2 Basic Operations

From the axioms set in subsection 9.1 further operations can be derived. As quaternion numbers are non-communative, these can differ from the $\mathbb{R}$ space. It is very common to multiply two quaternions. This operation is equal to

$$\begin{gathered} q_1 \cdot q_2 = (a_1 + b_1 i + c_1 j + d_1 k)(a_2 + b_2 i + c_2 j + d_2 k) \\ = (a_1 a_2 - b_1 b_2 - c_1 c_2 - d_1 d_2) + (a_1 b_2 + b_1 a_2 + c_1 d_2 - d_1 c_2) i \\ +(a_1 c_2 - b_1 d_2 + c_1 a_2 + d_1 b_2) j + (a_1 d_2 + b_1 c_2 - c_1 b_2 + d_1 a_2) k \end{gathered}$$

but can also be denoted as a matrix multiplication due to its complexity:

$$q_1 \cdot q_2 = \begin{pmatrix} a_2 & -b_2 & -c_2 & -d_2 \\ b_2 & a_2 & d_2 & -c_2 \\ c_2 & -d_2 & a_2 & b_2 \\ d_2 & c_2 & -b_2 & a_2 \end{pmatrix} \cdot \begin{pmatrix} a_1 \\ b_1 \\ c_1 \\ d_1 \end{pmatrix} \cdot \begin{pmatrix} 1 & i & j & k \end{pmatrix}.$$

Division makes use of the fact that $q_1/q_2 = q_1 \cdot q_2^{-1}$. The inverse of q corresponds to [16]

$$q^{-1} = \frac{\bar{q}}{|q|^2}.$$

This equation follows from:

$$q \cdot q^{-1} = 1 = \frac{|q|^2}{|q|^2} = q \frac{\bar{q}}{|q|^2}.$$

Lastly, the exponential of a quaternion e^q shares some similarities with Euler's formula [10] and can be written as

$$e^q = e^v (\cos |w| + \frac{w}{|w|} \sin |w|)$$

where $\mathrm{Sc}(q) = v$ and $\mathrm{Vec}(q) = w$ [17]. This follows from the general definition [17]

$$e^x = \sum_{k=0}^{\infty} \frac{x^k}{k!}.$$

The equation must hold true as for $\mathrm{Sc}(q) = v$ and $\mathrm{Vec}(q) = w$, $e^q = e^v \cdot e^w$. Furthermore, since w is a pure unit quaternion and thus $w^2 = (bi + cj + dk)^2 = -b^2 - c^2 - d^= - |w|^2$,

$$e^w = \sum_{k=0}^{\infty} \frac{w^k}{k!} = 1 + \frac{w}{1!} - \frac{|w|^2}{2!} - \frac{|w|^2 w}{3!} + \frac{|w|^4}{4!} + \cdots$$

These summands can then be divided into two groups which equal the Taylor series of cos and sin:

$$e^w = (1 - \frac{|w|^2}{2!} \frac{|w|^4}{4!} + \cdots) + \frac{w}{|w|}(\frac{|w|}{1!} - \frac{|w|^3}{3!} + \frac{|w|^5}{5!} + \cdots) = \cos(|w|) + \frac{w}{|w|}\sin(|w|).$$

This lastly gives

$$e^q = e^v \cdot e^w = e^v(\cos(|w|) + \frac{w}{|w|}\sin(|w|)).$$

10 Tracing Three-Dimensional Paths

Once again, three-dimensional paths will be approximated through a set of characteristic points that are determined through user input. There are two options to apply the Discrete Fourier Transform to such data. As in section 3, the index can be used to store a third component. The issues that this brings about have previously been discussed. A more sustainable solution is to, as seen in section 3, expand the input space. Complex numbers limit the input to two dimensions. Quaternion numbers represent an expansion of the space into four dimensions which allows an input of the same size. Once again a projection $\phi : \mathbb{R}^3 \to \mathbb{H}$ is defined which converts a point (x, y, z) to a quaternion $0 + xi + yj + zk$. The real dimension will remain unpopulated for now. Options to fill this spot will be discussed in subsection 11.4.

The step from $\mathbb{C}$ to $\mathbb{H}$, however, is not quite as a straight-forward as from $\mathbb{R}$ to $\mathbb{C}$. In the form that the DFT has been used thusfar it is uncapable of handling a quaternion input. It is altered, giving the Discrete Quaternion Fourier Transform or DQFT. Due to the lack of commutativity in the set of quaternion numbers, there are two such transforms: the right sided (RDQFT) and the left sided Discrete Quaternion Fourier Transform (LDQFT). The RDQFT is defined as [7]

$$X(f) = \sum_{n=0}^{N-1} x_n \cdot e^{-\mu 2\pi n f \frac{1}{N}}$$

while the LDQFT is equal to [7]

$$X(f) = \sum_{n=0}^{N-1} e^{-\mu 2\pi n f \frac{1}{N}} \cdot x_n.$$

The terms "left sided" and "right sided" refer to the position of the exponential function $e^{-\mu 2\pi n k \frac{1}{N}}$. This property will play an important role when choosing the inverse transform. The two are identical besides this factor in usage and results. As the RDQFT more closely resembles the DFT used so far, this project will solely rely on it and ignore the left sided transform. From now on the RDQFT will also be called the DQFT. Nonetheless, all findings apply to both. The inverse of the RDQFT is the following [7]:

$$x(f) = \frac{1}{N} \sum_{n=0}^{N-1} e^{\mu 2\pi n f \frac{1}{N}} \cdot X_n.$$

It will be abbreviated as the IDQFT. The transform and its inverse bear a close resemblance to their non-quaternionic counterparts. What sets them apart is that e has a quaternionic instead of a complex exponent. μ is a place-holder for any pure unit quaternion. This is a quaternion of length one that determines a direction in space. Throughout this project l has been chosen to equal μ in most cases.

11 How Do the DQFT and IDQFT Work?

11.1 Elliptical Epicycles

The IDQFT displayed in the ijk-space can vary from a traditional epicycle under certain conditions. Instead of being made up of many circles, it consists of many ellipses. This can be shown by taking a closer look at what the operation $e^{\mu 2\pi nf\frac{1}{N}} \cdot q$ where q is a quaternion expresses. First, μ will be picked to equal i. The mentioned multipication is thus equal to

$$e^{\mu 2\pi nf\frac{1}{N}} \cdot q = (\cos(\omega)a - \sin(\omega)b) + (\cos(\omega)b + \sin(\omega)a)i + (\cos(\omega)c - \sin(\omega)d)j + (\cos(\omega)d + \sin(\omega)c)k$$

with $\omega = \mu 2\pi nf\frac{1}{N}$. This in turn gives, when excluding the real dimension,

$$(\cos(\omega)b + \sin(\omega)a)i + (c + di)(\cos(\omega)j + \sin(\omega)k).$$

The multiplication thus expresses a circle on the jk-plane of radius $\sqrt{c^2 + d^2}$ that is shifted according to $(\cos(\omega)b + \sin(\omega)a)i$. This produces an ellipse as can be seen in figure 24. Such shapes can be observed no matter which dimension is left out, as there will always be a pair that forms such a circle. It is important to note that the circular base is independant of the values of a and b.

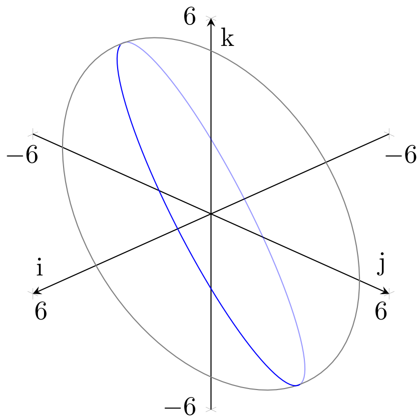


Figure 24: a geometric representation of $e^{i2\pi nf\frac{1}{N}} \cdot (0 + 5i + 4j + 3k)$

When μ equals j, a slight change can be seen. The multiplication then gives:

$$e^{\mu 2\pi nf\frac{1}{N}} \cdot q = (\cos(\omega)a - \sin(\omega)c) + (\cos(\omega)b + \sin(\omega)d)i + (\cos(\omega)c + \sin(\omega)a)j + (\cos(\omega)d - \sin(\omega)b)k$$

which is equal to

$$(\cos(\omega)c + \sin(\omega)a)j + (d + bj)(\cos(\omega)k + \sin(\omega)i)$$

if the real dimension is eliminated. As shown in figure 25, this represents a circle on the ik-plane that is stretched along the j-axis. Most importantly, the multiplication no longer runs through the same values. However, as will be shown in the next passage, this does not effect whether the IDQFT runs through the given data points or not. Lastly, when μ is equal to k the circular base moves to the ij-plane. In the case of whole number pure unit quaternions, the base is always located on the plane perpendicular to the direction vector that runs through the origin and q in ijk-space.

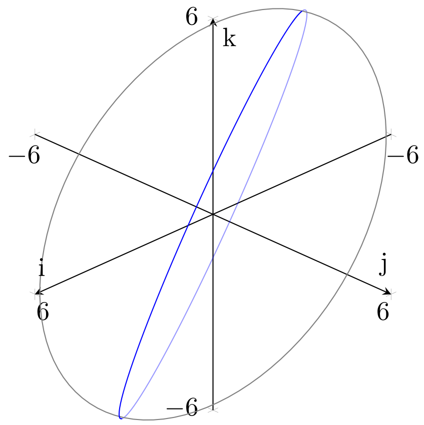


Figure 25: a geometric representation of $e^{i2\pi nf\frac{1}{N}} \cdot (0 + 5i + 4j + 3k)$

11.2 A Proof of the DQFT

This extract proves that the DQFT is capable of filtering out the coefficients X_n from a set of data. It bears a close resemblance to section 5 where the same has been shown for the DFT. The goal of the DQFT is to find the values X_n which allow the values x_n to be calculated through

$$x(f) = \frac{1}{N}\sum_{n=0}^{N-1} e^{\mu 2\pi nf\frac{1}{N}} X_n.$$

Since it can be assumed that an IDQFT can be found for all sets of values x_n, it can be inserted into the DQFT:

$$\sum_{n=0}^{N-1} e^{-\mu 2\pi nf\frac{1}{N}} x_n = \frac{1}{N}\sum_{n=0}^{N-1} e^{-\mu 2\pi\frac{nf}{N}}(\sum_{m=0}^{N-1} e^{\mu 2\pi\frac{mn}{N}} X_m) = \frac{1}{N}\sum_{n=0}^{N-1}(\sum_{m=0}^{N-1} e^{\mu 2\pi\frac{(m-f)n}{N}} X_m).$$

When $m = f$, the multiplication returns X_f. In order to show that the remaining summands for which $m \neq f$ sum up to 0, the equation is further transformed:

$$\frac{1}{N}\sum_{n=0}^{N-1}(\sum_{m=0}^{N-1} e^{\mu 2\pi\frac{(m-f)n}{N}} X_m) = \frac{1}{N}\sum_{m=0}^{N-1}(\sum_{n=0}^{N-1} e^{\mu 2\pi\frac{(m-f)n}{N}} X_m). \tag{4}$$

This shows that the inner sum defines a geometric series when $m \neq k$. From this follows that the geometric sum formula [11] can be applied:

$$\sum_{n=0}^{N-1} e^{\mu 2\pi\frac{(m-f)n}{N}} X_m = \sum_{n=1}^{N} e^{\mu 2\pi\frac{(m-f)(n-1)}{N}} X_m = X_m \frac{1 - e^{\mu 2\pi\frac{(m-f)N}{N}}}{1 - e^{\mu 2\pi\frac{m-f}{N}}} = X_m \frac{1 - e^{\mu 2\pi(m-f)}}{1 - e^{\mu 2\pi\frac{m-f}{N}}}$$

As μ is a pure unit quaternion, $e^{\mu 2\pi(m-f)}$ is equal to

$$e^v(\cos(|w|) + \frac{w}{|w|}\sin(|w|)) = e^v(\cos(2\pi(m-f)) + \mu\sin(2\pi(m-f))) = e^0(1+0) = 1$$

with $v = \text{Sc}(\mu 2\pi(m-f)) = 0$ and $w = \text{Vec}(\mu 2\pi(m-f))$. This implies

$$X_m \frac{1 - e^{\mu 2\pi(m-f)}}{1 - e^{\mu 2\pi \frac{m-f}{N}}} = X_m \frac{0}{1 - e^{\mu 2\pi \frac{m-f}{N}}} = 0.$$

This information can then be plugged into equation 4:

$$\frac{1}{N} \sum_{m=0}^{N-1} (\sum_{n=0}^{N-1} e^{\mu 2\pi \frac{(m-f)n}{N}} X_m) = \frac{1}{N} \sum_{m=0}^{N-1} X_f = X_f.$$

It has thus been shown that the DQFT can in fact extract the coefficients X_n from a set of values x_n.

11.3 Example

How one must go about when using the DQFT will be demonstrated in this subsection. The set of data used for this example is given in table 4. Figure 26 shows the points plotted in three-dimensional space along with their orthogonal projections onto the ij-plane. There are four points, implying that $N = 4$. In this example μ has chosen to equal k.

n	pts.	x_n
0	(4.619, 1.148, 2.613)	$4.619i + 1.148j + 2.613k$
1	(-1.913, 2.772, 1.082)	$-1.913i + 2.772j + 1.082k$
2	(-4.619, -1.148, -2.613)	$-4.619i - 1.148j - 2.613k$
3	(1.913, -2.772, -1.082)	$1.913i - 2.772j - 1.082k$

Table 4: an example set of three-dimensional data

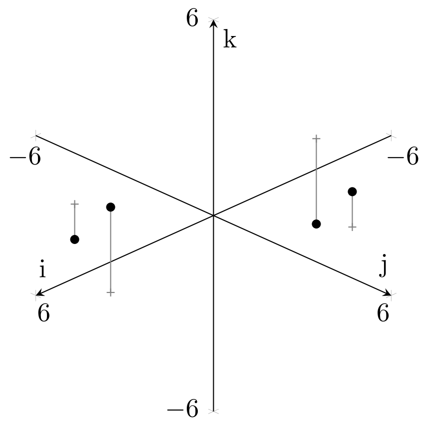


Figure 26: a plot of an example set of data

The first step is to calculate X_0. It is equal to

$$X_0 = (4.619i + 1.148j + 2.613k) + (-1.913i + 2.772j + 1.082k)$$
$$+(-4.619i - 1.148j - 2.613k) + (1.913i - 2.772j - 1.082k) = 0.$$

This value can already be determined by just plotting the values as in figure 26. It is clear that they are all equidistant from the origin, implying that the fixed point that the arrows will be connected to is also located there. In the next step X_1 is found to have a value of $1.082 + 7.391i + 3.061j + 2.613k$:

$$\begin{aligned} X_1 &= (4.619i + 1.148j + 2.613k)e^{-k0\frac{2\pi}{4}} + \cdots + (1.913i - 2.772j - 1.082k)e^{-k3\frac{2\pi}{4}} \\ &= (4.619i + 1.148j + 2.613k) + \cdots + (1.913i - 2.772j - 1.082k)(\cos(3\frac{2\pi}{4}) - k\sin(3\frac{2\pi}{4})) \\ &= 2.164 + 14.782i + 6.122j + 5.226k. \end{aligned}$$

The remaining coefficients are $X_2 = 0$ and $X_3 = -2.164 + 3.694i - 1.530j + 5.226k$. With these values the IDQFT has been determined:

$$x(f) = \frac{1}{4}(2.164 + 14.782i + 6.122j + 5.226k)e^{k\frac{2\pi f}{4}} + \frac{1}{4}(-2.164 + 3.694i - 1.530j + 5.226k)e^{k3\frac{2\pi f}{4}}.$$

It can be confirmed that this in fact holds true for x_0, x_1, x_2 and x_3. The path taken by the IDQFT has additionally been plotted in figure 27. Alongside this, the elliptical interpretation of the transform is shown.

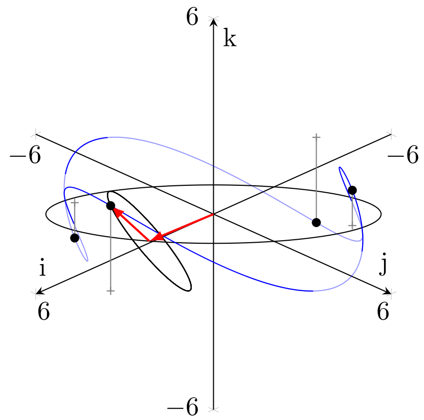


Figure 27: the IDQFT of an example set of data

11.4 Representation of the Fourth Dimension

Since our world is limited to three spacial dimensions, the representation of a fourth spacial axis is rather difficult. For this reason other mediums are often chosen. Points in space are most commonly visualized through dots. This allows the communication of a fourth value through their size or shape. Unfortunately, such methods are often misleading and create clutter. Sound can also be used in certain circumstances but has no general applications. In these situations every value is matched with a certain pitch.

It is much more popular to instead change the color of respective coordinates. For example, a black dot could correspond to the value ten while a white dot could equal zero. Colors further have the advantage that they can be defined through a wide range of values. They can be described as warm/cold, dark/light, or even appealing/unappealing. Such properties, however, are difficult to assign concrete values to and thus are unsuited. The wavelength of a color, on the other hand, is far more fittng as it allows a color to be uniquely identified through a single value. While this solution can be easily understood, it is not commonly used due to the various calculations that are involved and limited domain.

As computers often use the RGB or HSL color models these are by far the most convenient. The RGB format consists of three single values that range from 0 to 255 [18]. Each represents the amount red, green, or blue present in a color. This allows the creation of a linear interpolation similar to the following between two colors (r_1, g_1, b_1) and (r_2, g_2, b_2):

$$r(x) = \frac{x}{x_{max}} \cdot \Delta r + r_1, \quad g(x) = \frac{x}{x_{max}} \cdot \Delta g + g_1, \quad b(x) = \frac{x}{x_{max}} \cdot \Delta b + b_1$$

where $x \in [0, x_{max}]$ and $\Delta r = r_2 - r_1$, $\Delta g = g_2 - g_1$, and $\Delta b = b_2 - b_1$. The domain of x must be determined beforehand. A Fourier Transform making use of such a scale where red equals six and blue negative six can be found in figure 28. Similar calculations can be made for the HSL mode where $H \in [0°, 360°]$, $S_L \in [0, 1]$, and $L \in [0, 1]$ [18].

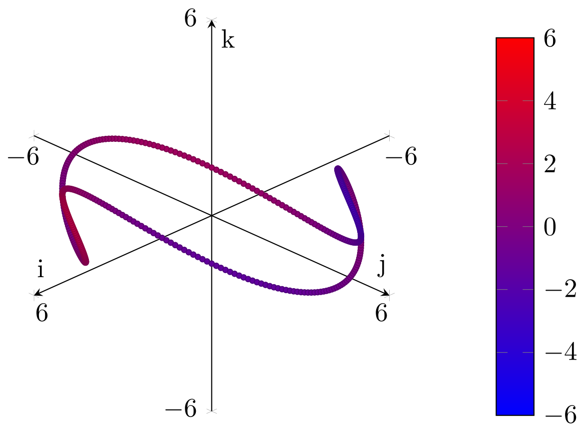


Figure 28: a set of data in which the fourth dimension is visualized through color

12 Automization of the DQFT and IDQFT

In addition to a program that demonstrates the DFT, a piece of software has been created that presents the DQFT. It has been coded in JavaScript as well and can be found at **dqft.birmanns.org**. The exact code is located in appendix B. A number of screenshots and examples can be found in section 13. They feature the program itself and animations it has created.

12.1 Usage

To the right side of the screen the user can find fields to enter the x, y, and z coordinates of one of their desired points. Since it is rather difficult to use a mouse or touch screen to draw a three-dimensional path, this method must be used instead of allowing the user to create them through motion. Once the information has been entered, it can then be added to the space through the plus button. The individual axis are limited to a domain of 0 to 20, coordinates outside of this range cannot be entered. At the center of the screen the isometric projection of an empty space is shown. It consists of just three axis that represent a quaternion space after removing the real dimension. As more and more points are added the space fills with crosses located at the corresponding spots. A simple projection $\phi : \mathbb{R}^3 \to \mathbb{H}$ is used here that transforms a point (x, y, z) to a quaternion $xi + yj + zk$. The slider located at the bottom of the screen can be used to turn the scene around the k-axis. Below the slider one can choose whether to show or hide the fourth dimension. It is represented through a range of colors and based on a linear interpolation between a shade of yellow and blue. This method has previously been described in subsection 11.4.

Once two or more points have been added, a chain of arrows will start tracing a shape that connects them. The IDQFT is used for this with $\mu = k$. This implies that arrows will appear to constantly change their length unless viewed such that the k-axis disappears. They follow an elliptical path that has previously been described in section 11.1. The last arrow's tip is followed by a trail that traces back $N - 1$ points. Conventional computers will experience performance issues as soon as seven or more coordinates have been added. For this reason the user is prevented from adding more than six. They in turn have the option to remove previously added points or alter the order that they are being traced in.

12.2 Rendering

The simple three-dimensional effect is achieved through a series of matrix multiplications. The order the steps are completed in is of high importance as they are non-commutative. In the first step the single points are rotated around the k-axis through the following multiplicaiton:

$$\begin{pmatrix} \cos(\alpha) & -\sin(\alpha) & 0 \\ \sin(\alpha) & \cos(\alpha) & 0 \\ 0 & 0 & 1 \end{pmatrix} \cdot \vec{p}_3 = \vec{p}_{3,r}$$

where α equals the current angle of the i-axis to its original position and $\vec{p}_3$ the vector from the origin to the specific coordinates of a point. In the second step it is translated from the three-dimensional space to a two-dimensional plane through an isometric projection. This is done through the following matrix multiplication:

$$\begin{pmatrix} -\frac{\sqrt{3}}{2} & \frac{\sqrt{3}}{2} & 0 \\ -\frac{1}{2} & -\frac{1}{2} & 1 \end{pmatrix} \cdot \vec{p}_{3,4} = \vec{p}_2.$$

In a last step the vector is scaled and translated to fit the window. Once completed, one is left with a vector that is equivalent to the coordinates of the given point on the screen.

12.3 Further Development

In its current state the program already completes the tasks it was set to achieve, nonetheless, there are features that could improve the experience. In its current version the user is restricted in their viewing experience. A second dimension of movement could enable them to further understand the process displayed. Especially an option of viewing the arrows from directly above could prove beneficial. It would allow the ellipses to appear as cricles as the k-axis disappears and only the ij-plane is visible. Before this can be achieved, however, the program's performance must be improved. This would also make the addition of further features possible. Most importantly, the ability to add more points could be implemented and thus more complex preset examples.

13 Examples in Three-Dimensional Space

As an addition to section 12, this one will present screen shots and videos from the software that has been created. It is recommended that one also visits **dqft.birmanns.org**. Every screenshot has been matched with a qr-code that leads to the video that the image stems from. The first sample presents the IDQFT as it connects five randomly chosen points.

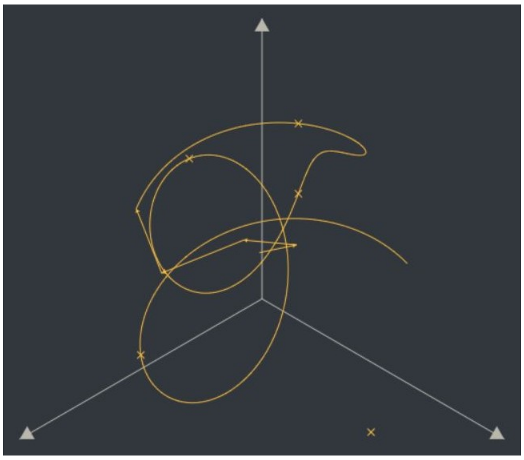


Figure 29: a screenshot of an IDQFT tracing five random points

Figure 30: https://youtu.be/PClDqjzHCLM

In the second example four random points are added. Subsequently, the scene is rotated back and forth, presenting the epicycle from all sides.

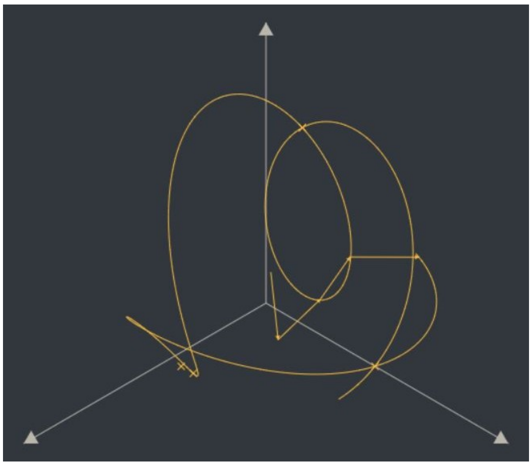


Figure 31: a screenshot of an IDQFT tracing four random points

Figure 32: https://youtu.be/1-M3gxb9zYo

The last presents a four-dimensional interpretation of the Fourier Transform. Color has been chosen as a fourth axis. It interpolates linearly from (0,218,255) to (176,126,26). The set of data is made up of four random points consisting of four values each.

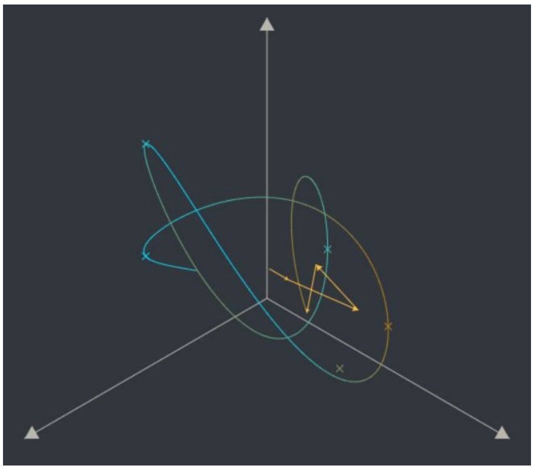


Figure 33: a screenshot of an IDQFT tracing four random four-dimensional points

Figure 34: https://youtu.be/LTKy-dPIYOo

14 Concluding Remarks

As is the case for all of mathematics, Fourier Anaylsis is a field that seems to have no limits. With every discovery, many more unknowns are uncovered. It is for this reason that boundaries but also goals must be set. The moment of completing these has been reached in this paper. The phenomenon that prompted this project has been explained and elaborated on. Both the Discrete Fourier Transform and Discrete Quaternion Fourier Transform have been discussed in much detail along with their domains, the sets of complex and quaternion numbers. These transforms had previously only been discussed briefly in scientific resources accessible to the target audience.

The first step was taken by demonstrating that the DFT is capable of tracing drawings through the help of the complex plane. It was accordingly proven that the IDFT can be understood as a series of arrows or an epicycle. From this set of theory a piece of software could be developed that presents the visual appeal that the Fourier Transform can have as well. A similar strategy was followed in the three- and four-dimensional space. The DQFT and IDQFT were shown to have the ability to follow three-dimensional paths. After finding a proof for this transform a short discussion about visualizing a fourth dimension ensued. A second piece of software was developed to present this theory as well.

There are a range of questions that have also been chosen to remain unanswered. Some have already been named in subsections 7.3 and 12.3. Further, as the two-, three-, and four-dimensional spaces have been explored, the next step would be the research of the five- or even n-dimensional spaces. Many more pieces of software could be developped as well. The project has limited the number of dimensions due to the given time frame. Various areas of Fourier Analysis and a number of transforms have also remained unnamed for the same reason.

References

[1] Ronald Newbold Bracewell. *The Fourier transform and its applications*, volume 31999. McGraw-Hill New York, 1986.

[2] M Richardson. Fundamentals of the discrete fourier transform. *Sound & Vibration Magazine*, page 5, 1978.

[3] Grant Sanderson. But what is a fourier series? from heat flow to drawing with circles. http://youtu.be/r6sGWTCMz2k, Jun 2019.

[4] Fourier transforms & the frequency domain. w.astro.berkeley.edu/ jrg/ngst/fft/applicns.html.

[5] Rohit Thummalapalli. Fourier transform: Nature's way of analyzing data. *Yale Scientific*, 2010.

[6] PhD Christian Heinrich Friedrich Peters and Edward Ball Knobel. *Ptolemy's Catalogue of Stars*. The Carnegie Institution of Washington, 1915.

[7] Ben Kenwright. Quaternion fourier transform for character motions. *Eurographics Digital Library*, page 2, 2015.

[8] Douglas Harper. Etymology of epicycle. *Online Etymology Dictionary*, 2021.

[9] Bradly Carrol and Dale Ostlie. *An Introduction to Modern Astrophysics*. Addison-Wesley, 2 edition, 2007.

[10] Nayana Nair. Euler's identity. *Éclat Mathematics Journal*, 10:2, 2018–2019.

[11] Werner Durandi, Baoswan Dzung Wong, Markus Kriener, Hansruedi Künsch, Alfred Vogelsanger, Jörg Waldvogel, Samuel Byland, Klemens Koch, Andreas Bartlome, Michael Bleichenbacher, and Hans Roth. *Formeln, Tabellen, Begriffe*. Orell Füssli Verlag, 6 edition, 2017.

[12] Frank A Farris. Wheels on wheels on wheels — surprising symmetry. *Mathematics Magazine*, 69(3):185–189, 1996.

[13] Khalid Suleiman Al-Kharabsheh, Ibrahim Mahmoud AlTurani, Abdallah Mahmoud Ibrahim AlTurani, and Nabeel Imhammed Zanoon. Review on sorting algorithms a comparative study. *International Journal of Computer Science and Security (IJCSS)*, 7(3):120–126, 2013.

[14] Eric W Weisstein. Euler angles. *https://mathworld.wolfram.com)*, 2009.

[15] William L. Hosch. quaternion. https://www.britannica.com/science/quaternion, Aug 2009.

[16] Mawardi Bahri. Discrete quaternion fourier transform and properties. *Inj. J. Math Analysis*, 7(25):1208, 2013.

[17] Dong Cheng and Kit Ian Kou. Plancherel theorem and quaternion fourier transform for square integrable functions. *Complex Variables and Elliptic Equations*, 64(2):223–242, Jan 2018.

[18] Martin Loesdau, Sébastien Chabrier, and Alban Gabillon. *Hue and Saturation in the RGB Color Space*. Springer International Publishing, Cham, 2014.

Appendix A Listing

Visualization DFT

Visualization DQFT

Appendix B Source Code Visualization DFT

The code for the program that visualizes the DFT consists of multiple documents. Their contents can be found in the following listings.

Listing 1: index.html

```html
<!DOCTYPE html>
<html lang="en">

<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <meta http-equiv="X-UA-Compatible" content="ie=edge">
  <title> Complex Fourier Transform </title>

  <!-- CSS file -->
  <link href="styles.css" rel="stylesheet">
  <!-- animation library Anime.js -->
  <script src="./anime-master/lib/anime.min.js"></script>
  <!-- preloads UI transitions -->
  <script defer type ="module" src="./UI.js"></script>
  <!-- main script -->
  <script defer src="script.js" type="module"></script>
</head>

<body>
  <!-- canvases -->
  <div id="wrapper">
    <canvas id="detection_canvas"></canvas>
    <canvas id="drawing_canvas"></canvas>
    <canvas id="points_canvas"></canvas>
    <canvas id="arrows_canvas"></canvas>
  </div>

  <!-- button positioned at the bottom center of the screen -->
  <button id="button_main">
    <!-- gets the user's arrow number input -->
    <input id="arrow_number_input" type="number"></input>

    <!-- calculation text -->
    <div id="calculation_text">Run Calculation</div>

    <!-- play/pause graphics -->
    <svg id="play-pause" width="35" height="35" viewBox="0 0 35 35" >
      <path id="pp_path0" d="M0 0H15V35H0V0Z" fill="#000000"/>
      <path id="pp_path1" d="M20 0H35V35H20V0Z" fill="#000000"/>
    </svg>
  </button>

  <!-- drawer that presents various examples -->
  <div id="drawer_examples" data-toggled="false">
    <div id="header_wrapper">
      Examples
      <div id="arrow"></div>
    </div>
    <!-- description of all examples -->
    <div id="examples_wrapper">
      <div data-source="KSimLee.txt">KS Im Lee</div>
      <div data-source="PI.txt">PI</div>
    </div>
  </div>

  <!-- further buttons -->
  <button id="button_reset"></button>
  <button id="button_restart"></button>
  <button id="button_confirm">Confirm</button>

</body>

</html>
```

Listing 2: styles.css

```css
* {
  /* appearance */
  margin: 0;
  padding: 0;
  box-sizing: border-box;

  /* font */
  font-family: Helvetica;
  font-color: #27292b;
}

body {
```

```
  /* apperance */
  overflow: hidden;
  background-color: #32373e;
}




#detection_canvas {
  /* position */
  position: absolute;
  z-index: -1;
}

#drawing_canvas {
  /* position */
  position: absolute;
  z-index: -4;
}

#points_canvas {
  /* position */
  position: absolute;
  z-index: -3;
}

#arrows_canvas {
  /* position */
  position: absolute;
  z-index: -2;
}



#button_main {
  /* position */
  position: fixed;
  left: 50%;
  bottom: 50px;
  transform: translate(-50%,50%);
  z-index: 4;

  /* appearance */
  width: 280px;
  height: 60px;
  border: none;
  border-radius: 30px;
  background-color: #f2b25c;

  /* font */
  text-align: center;
  color: black;
  text-decoration: none;
  font-size: 30px;

  /* misc */
  cursor: pointer;
}

#calculation_text{
  /* position */
  position: absolute;
  top: 0;

  /* appearance */
  width: 100%;
  height: 100%;

  /* children */
  line-height: 60px;
  text-align: center;
}

#input_wrapper{
  /* appearance */
  display: table-cell;
  width:100%;
  height:100%;

  /* children */
  align-items: center;
  vertical-align: middle;
  line-height: 60px;
}

#arrow_number_input{
  /* position */
  position: relative;
  z-index: 1;

  /* appearance */
  display: none;
```

```
  width: 80px;
  height: 40px;
  opacity: 0;
  border-width: 0;
  border-radius: 5px;
  background-color: #de9f57;

  /* font */
  text-align: center;
  font-size: 30px;

  /* misc */
  cursor: text;
}

#arrow_number_input:focus{
  /* appearance */
  outline-color: #ac8146;
}

/* removes up and down arrows from number input */
#arrow_number_input::-webkit-outer-spin-button, #arrow_number_input::-webkit-inner-spin-button {
  /* appearance */
  -webkit-appearance: none;
  margin: 0;
}

#play-pause {
  /* position */
  position: absolute;
  right: 12.5px;
  bottom: 12.5px;

  /* appearance */
  display: none;
  opacity: 0;
}



#drawer_examples {
  /* position */
  position: fixed;
  left: 50%;
  bottom: 70px;
  transform: translate(-50%,0);
  z-index: 2;

  /* appearance */
  width: 210px;
  height: 35px;
  padding: 5px;
  background-color: #4c4e50;
  border-radius: 10px;
  text-align: center;
  overflow: scroll;
}

#drawer_examples::-webkit-scrollbar {
  /* appearance */
  display: none;
}

#examples_wrapper > * {
  /* appearance */
  width: 100%;
  --height: 3.4vh;
  border-radius: calc(var(--height)*0.5);
  background-color: #5b5d60;
  margin-top: 5px;

  /* children */
  line-height: var(--height);

  /* misc */
  cursor: pointer;
}

#header_wrapper{
  /* apperance */
  width:100%;
  height:30px; /* is changed in UI.openDrawer() */

  /* children */
  text-align: center;
}

#arrow {
  /* position */
  position: relative;
  left: 24%;
  top: -80%;
```

```

  /* apperance */
  width: 0;
  height: 0;
  border-left: 7px solid transparent;
  border-right: 7px solid transparent;
  border-bottom: 7px solid #27292b;
  margin: auto;
  margin-bottom: 10px;

  /* misc */
  cursor: pointer;
}



#button_reset{
  /* position */
  position: fixed;
  left: calc(50% + 190px);
  bottom: 50px;
  transform: translate(-50%,50%);
  z-index: 3;

  /* appearance */
  height: 60px;
  width: 60px;
  border: none;
  border-radius: 30px;
  background-color: #f06d65;

  /* font */
  font-size: 20px;

  /* misc */
  cursor: pointer;
}

#button_restart{
  /* position */
  position: fixed;
  bottom: 50px;
  left: 50%;
  transform: translate(-50%,50%);
  z-index: 3;

  /* appearance */
  height: 45px;
  width: 45px;
  opacity:0;
  border: none;
  border-radius: 50%;
  background-color: #f06d65;

  /* font */
  font-size: 20px;

  /* misc */
  cursor: pointer;
}

#button_confirm{
  /* position */
  position: fixed;
  left: 50%;
  bottom: 33px;
  transform: translate(-50%,50%);
  z-index: 10;

  /* apperance */
  display: none;
  height: 30px;
  width: 90px;
  opacity: 0;
  border: none;
  border-radius: 15px;
  background-color: #56c2b8;

  /* font */
  font-size: 20px;

  /* misc */
  cursor: pointer;
}
```

Listing 3: script.js

```
//Import modules
import * as draw from "./draw.js";
import * as calc from "./calculation.js";
import * as UI from "./UI.js";
import { arrowTracer } from "./arrowTracerClass.js";


//Canvases
//Detects movement
const detectionCanvas = document.querySelector("#detection_canvas");
//Shows drawing
const drawingCanvas = document.querySelector("#drawing_canvas");
const ctxDrw = drawingCanvas.getContext('2d');
//Shows drawing as individual points
const pointsCanvas = document.querySelector("#points_canvas");
const ctxPts = pointsCanvas.getContext('2d');
//Contains moving arrows
const arrowsCanvas = document.querySelector("#arrows_canvas");
const ctxArrows = arrowsCanvas.getContext('2d');
//Array containing all canvases
const canvasList = ["#detection_canvas","#drawing_canvas","#points_canvas","#arrows_canvas"];


//UI
const buttonMain = document.querySelector("#button_main");
const drawerExamplesDiv = document.querySelector("#drawer_examples");
const toggleArrow = document.querySelector("#arrow");
const arrowInput = document.querySelector("#arrow_number_input");
const buttonConfirm = document.querySelector("#button_confirm");
const buttonReset = document.querySelector("#button_reset");
const examplesWrapper = document.querySelector("#examples_wrapper");
const buttonRestart = document.querySelector("#button_restart");


//Tracks the center of the screen
let origin = [window.innerWidth/2,window.innerHeight/2];

//Drawing variables
let drawing = false;
let coloredPixels = [];


//Contains current state
let currentState = 0;
//0: drawing phase
//1: calculation settings phase
//2: output phase (play)
//3: output phase (pause)




//* DRAWING *//


//Sets various properties once the program is loaded
window.addEventListener('load', () => {
  ctxDrw.lineWidth = 3;
  resizeWindow();
  //Load examples
  UI.setProperties();

  //Show canvas that displays drawing
  drawingCanvas.style.display = "block";
  //Hide canvas that displays drawing as individual crosses
  pointsCanvas.style.display = "none";
});


//Starts drawing when the mouse is pressed down
detectionCanvas.addEventListener('mousedown', () => {
  if(currentState == 0){
  //Get mouse position
  let mousePositionX = window.event.pageX;
  let mousePositionY = window.event.pageY;

  //Start drawing
  drawing = true;
  ctxDrw.moveTo(mousePositionX, mousePositionY);
  ctxDrw.beginPath();
  }
})


//Ends drawing when the mouse is lifted
detectionCanvas.addEventListener('mouseup', () => {
  drawing = false;
  ctxDrw.closePath();
})

```

```

//Ends drawing if the mouse leaves the window
detectionCanvas.addEventListener('mouseout', () => {
  drawing = false;
})


//Draws a line to the new mouse position when it is moved and drawing is activated
detectionCanvas.addEventListener('mousemove', () => {
  if(drawing){
    //Gets the new mouse position
    let mousePositionX = window.event.pageX;
    let mousePositionY = window.event.pageY;

    //Draws the line
    ctxDrw.strokeStyle = "#BAB7AC";
    ctxDrw.lineTo(mousePositionX, mousePositionY);
    ctxDrw.stroke();
    coloredPixels.push([mousePositionX, mousePositionY]);

    //Adds a cross
    draw.drawCross(ctxPts, [mousePositionX, mousePositionY], 5, "#BAB7AC");
  }
})


//Starts drawing when a touch is detected
detectionCanvas.addEventListener('touchstart', () => {
  if(currentState == 0){
    //Get touch position
    let touchPositionX = event.touches[0].pageX;
    let touchPositionY = event.touches[0].pageY;

    //Start drawing
    drawing = true;
    ctxDrw.moveTo(touchPositionX, touchPositionY);
    ctxDrw.beginPath();
  }
})


//Ends drawing when the touch ends
detectionCanvas.addEventListener('touchend', () => {
  drawing = false;
  ctxDrw.closePath();
})


//Draws a line to the new touch position when it is moved and drawing is activated
detectionCanvas.addEventListener('touchmove', () => {
  if(drawing){
    //Gets the new touch position
    let touchPositionX = event.touches[0].pageX;
    let touchPositionY = event.touches[0].pageY;

    //Draws the line
    ctxDrw.strokeStyle = "#BAB7AC";
    ctxDrw.lineTo(touchPositionX, touchPositionY);
    ctxDrw.stroke();
    coloredPixels.push([touchPositionX, touchPositionY]);

    //Adds a cross
    draw.drawCross(ctxPts, [touchPositionX, touchPositionY], 5, "#BAB7AC");
  }
});




//* UI *//


window.addEventListener("resize", resizeWindow);


//Prevents refreshing through pulling down on Safari
if (window.safari) {
  history.pushState(null, null, location.href);
  window.onpopstate = function() {
      history.go(1);
  };
}


//Turns example divs into buttons
let examplesList = examplesWrapper.getElementsByTagName('div');
for(let i = 0; i < examplesList.length; i++){
  //Selects example as current drawing
  examplesList[i].addEventListener('click', () => {
    coloredPixels = [];
    //Loads values of the example from a txt-file
    getCoordinates(examplesList[i].dataset.source).then(function(result) {
```

```
      //Generate additional data
      coloredPixels = fillCoordinates(result);
      coloredPixels = fillCoordinates(coloredPixels);
      //Converts to next phase
      currentState = 1;
      UI.morphButtonMain(currentState);
      drawingCanvas.style.display = "none";
      pointsCanvas.style.display = "block";
      draw.drawCrosses(ctxPts, coloredPixels, 5, "#BAB7AC");
    })
  })
}


buttonMain.addEventListener('click', () => {
  if(currentState == 0 && coloredPixels.length > 0){
    //Convert to customization phase
    currentState = 1;
    UI.morphButtonMain(currentState);
    drawingCanvas.style.display = "none";
    pointsCanvas.style.display = "block";
  } else if(currentState == 2){
    //Pause animation
    window.cancelAnimationFrame(arrowAnim);
    currentState = 3;
    UI.togglePlayPause(0);
  } else if(currentState == 3){
    //Play animation
    runAnimation(testArrows);
    currentState = 2;
    UI.togglePlayPause(1);
  }
})


//Resets the drawing
buttonReset.addEventListener('click', () => {
  UI.resetCanvas(ctxDrw,drawingCanvas);
  UI.resetCanvas(ctxPts,pointsCanvas);
  coloredPixels = [];
})


//Loads an example arrow animation every time the arrow number is changed
let testArrows = "";
arrowInput.addEventListener('input', () => {
  if(arrowInput.value > coloredPixels.length){
    arrowInput.value = parseInt(coloredPixels.length);
  }
  testArrows = new arrowTracer(calc.c_bubbleSort(calc.c_dft(coloredPixels,parseInt(arrowInput.value/2))));
})


//Moves to phase 2 once the confirm button has been pressed
buttonConfirm.addEventListener('click', () => {
  if(currentState == 1 && arrowInput.value > 0){
    runAnimation(testArrows);
    drawingCanvas.style.display = "block";
    pointsCanvas.style.display = "none";
    currentState = 2;
    UI.morphButtonMain(currentState);
  }
})


//Completely resets the code when the restart button is pressed
buttonRestart.addEventListener('click', () => {
  window.cancelAnimationFrame(arrowAnim);
  window.cancelAnimationFrame(testArrows);
  UI.resetCanvas(ctxDrw,drawingCanvas);
  UI.resetCanvas(ctxPts,pointsCanvas);
  UI.resetCanvas(ctxArrows,arrowsCanvas);
  coloredPixels = [];
  currentState=0;
  UI.morphButtonMain(currentState);
  UI.togglePlayPause(1);
});


//Opens and closes examples drawer
toggleArrow.addEventListener('click', () => {
  if(drawerExamplesDiv.dataset.toggled == "true"){
    UI.closeDrawer();
  } else {
    UI.openDrawer();
  }
});




//* MISC *//
```

```


//Updates the arrow animation every 10ms
let arrowAnim;
function runAnimation(object){
  setTimeout(function(){
    if(currentState == 2){
      object.update();
      object.Frame += 0.003;
      arrowAnim = window.requestAnimationFrame(function(){runAnimation(object);});
    }
  }, 10);
}


//Loads values from a txt-file
async function getCoordinates(file){
  let result = [];
  await fetch(file).then(reponse => reponse.text()).then(text => {
    let lines = text.split("\r\n");
    for(let i = 0; i < lines.length -1; i++){
      let coordinates = lines[i].split(", ");
      //Converts relative positions to global positions
      let windowSize = [window.innerWidth, window.innerHeight];
      result.push([parseFloat(coordinates[0])+windowSize[0]/2,parseFloat(coordinates[1])+windowSize[1]/2]);
    }
  })
  return result;
}


//Adds the midpoint of every two adjacent points to a set of data
function fillCoordinates(coordinates){
  let result = [];
  for(let i = 0; i < coordinates.length; i++){
    result.push(coordinates[i]);
    let fillCord = [];
    fillCord[0] = (coordinates[i][0] + coordinates[(i+1)%coordinates.length][0]) / 2;
    fillCord[1] = (coordinates[i][1] + coordinates[(i+1)%coordinates.length][1]) / 2;
    result.push(fillCord);
  }
  return result;
}


//Makes various adjusments when window is resized
function resizeWindow() {

  //Determines points relative to origin before rescaling
  let relativePixels = [];
  for(let i=0; i < coloredPixels.length; i++){
    relativePixels.push([coloredPixels[i][0]-origin[0],coloredPixels[i][1]-origin[1]]);
  }

  //Updates the sizes of the cavases to match the screen
  //Automatically clears canvases
  for(let i = 0; i < canvasList.length; i++){
    let canvas = document.querySelector(canvasList[i]);
    canvas.height = window.innerHeight;
    canvas.width = window.innerWidth;
  }

  //Updates the position of the center of the screen
  origin = [window.innerWidth/2,window.innerHeight/2];

  for(let i=0; i<relativePixels.length; i++){
    coloredPixels[i][0]=relativePixels[i][0]+origin[0];
    coloredPixels[i][1]=relativePixels[i][1]+origin[1];
  }

  if(coloredPixels.length > 0){

    //Reset drawing process
    drawing = false;
    ctxDrw.closePath();

    //Recreates the drawing's path and crosses
    ctxDrw.strokeStyle = "#BAB7AC";
    ctxDrw.moveTo(coloredPixels[0][0],coloredPixels[0][1]);
    ctxDrw.beginPath();
    draw.drawCross(ctxPts, coloredPixels[0], 5, "#BAB7AC");

    for(let i=1; i< coloredPixels.length; i++){
      ctxDrw.lineTo(coloredPixels[i][0],coloredPixels[i][1]);
      ctxDrw.stroke();

      draw.drawCross(ctxPts, coloredPixels[i], 5, "#BAB7AC");
    }
    ctxDrw.closePath();

    //Restarts arrow preview animation to match new point positions
    if(currentState == 1){
```

```
      testArrows = new arrowTracer(calc.c_bubbleSort(calc.c_dft(coloredPixels,parseInt(arrowInput.value/2))))
    }

    //Resets arrow animation to match new point positions
    if(currentState > 1){
      window.cancelAnimationFrame(arrowAnim);
      testArrows = new arrowTracer(calc.c_bubbleSort(calc.c_dft(coloredPixels,parseInt(arrowInput.value/2))));
      runAnimation(testArrows);
    }
  }

}
```

Listing 4: calculation.js

```
//This file contains all functions related to calculations


//Calculates the length from the origin to a point / complex number
export function mgn(complex_number){
  let magnitude = Math.sqrt(Math.pow(complex_number[0],2)+Math.pow(complex_number[1],2));
  return magnitude
}


//Calculates the angle of a point / complex number to the origin
export function c_ang(complex_number){
  let angle = Math.atan(complex_number[1]/complex_number[0]);
  if(complex_number[0]<0){
    angle += Math.PI;
  } else if(complex_number[1]<0){
    angle += 2*Math.PI;
  }
  return angle;
}


//Sorts complex coefficients by magnitude, using the bubble sort method
export function c_bubbleSort(arr){
  var len = arr.length;
  var magnitudeArray = [];
  for(let j = 0; j < len; j++){
    let magnitude = mgn(arr[j][1]);
    magnitudeArray.push(magnitude);
  }
  for (var i = len-1; i>=0; i--){
    for(var j = 1; j<=i; j++){
      if(magnitudeArray[j-1]<magnitudeArray[j]){
        var temp = arr[j-1];
        arr[j-1] = arr[j];
        arr[j] = temp;
        temp = magnitudeArray[j-1];
        magnitudeArray[j-1] = magnitudeArray[j];
        magnitudeArray[j] = temp;
      }
    }
  }
  return arr;
}


//Performs a complex fourier transform up to the bin N
export function c_dft(values, N){
  let compoundResult = [];
  N=parseInt(N);
  for(let bin = -N+1; bin < N; bin++){
    let complex_result = [0,0];

    for(let i = 0; i < values.length; i++){
      complex_result[0] += values[i][0] * Math.cos(2 * Math.PI * bin * i / values.length);
      complex_result[0] += values[i][1] * Math.sin(2 * Math.PI * bin * i / values.length);
      complex_result[1] -= values[i][0] * Math.sin(2 * Math.PI * bin * i / values.length);
      complex_result[1] += values[i][1] * Math.cos(2 * Math.PI * bin * i / values.length);

    }
    compoundResult.push([bin,[complex_result[0]/values.length,complex_result[1]/values.length]]);
    // compoundResult.push([bin,[complex_result[0],complex_result[1]]]);
  }
  return compoundResult;
}


//The Inverse Discrete Fourier Transform
export function c_idft(coefficients, frame){
  let complex_result = [0,0];
  let N = coefficients.length;

  for(let k=0; k<coefficients.length; k++){
      complex_result[0] += coefficients[k][1][0] * Math.cos(-2 * Math.PI * coefficients[k][0] * frame/N);
      complex_result[0] -= coefficients[k][1][1] * Math.sin(-2 * Math.PI * coefficients[k][0] * frame/N);
```

```
      complex_result[1] += coefficients[k][1][0] * Math.sin(-2 * Math.PI * coefficients[k][0] * frame/N);
      complex_result[1] += coefficients[k][1][1] * Math.cos(-2 * Math.PI * coefficients[k][0] * frame/N);
  }

  return complex_result;
}
```

Listing 5: arrowTracerClass.js

```
//This file contains the arrowTracer class

import * as draw from "./draw.js";
import * as calc from "./calculation.js";

//Canvas that the arrowTracer class is drawn on
const arrowsCanvas = document.querySelector("#arrows_canvas");
const ctxArrows = arrowsCanvas.getContext('2d');


//Class that creates the spinning arrows
export class arrowTracer{

  //Varaible that holds the object
  set changeTracerObj(value){
    this.tracerObj = value;
  }
  get getTracerObj(){
    return this.tracerObj;
  }

  //Variable that holds the current frame
  set changeFrame(value){
    this.Frame = value;
  }
  get getFrame(){
    return this.Frame;
  }

  //The value the last arrow points at
  set changeCurrentVal(value){
    this.currentVal = value;
  }
  get getCurrentVal(){
    return this.currentVal;
  }

  //Keeps track of points the trail goes through
  set changeTrailLog(value){
    this.trailLog = value;
  }
  get getTrailLog(){
    return this.trailLog;
  }



  constructor(coefficients){
    //Sets variables to default values
    this.Frame = 0;
    this.coefficients = coefficients;
    this.trailLog = [];

    //Creates the object that contains the arrows
    //First creates a temporary place holder
    let tempObj = {};
    for(let i = 0; i < this.coefficients.length; i++){
      tempObj["arrow"+i.toString()] = {
        pointingTo: [0,0],
        length: calc.mgn(this.coefficients[i][1]),
        angle: calc.c_ang(this.coefficients[i][1]),
        frequency: this.coefficients[i][0]
      };
    }
    //Applys the temporary place holder to the actual object
    this.changeTracerObj = tempObj;

    // this.addSliders();
    this.update();
  }

  //Updates the arrow positions according to the current frame
  update(){
    //Clears the canvas
    ctxArrows.clearRect(0,0,arrowsCanvas.width,arrowsCanvas.height);

    //Draws the arrows according to the values store in the arrow object
    for(let i = 0; i < Object.keys(this.tracerObj).length; i++){
      //Extracts values from the arrow object
      let angle = this.tracerObj["arrow"+i.toString()].angle + this.Frame*this.tracerObj["arrow"+i.toString()].frequency;
      let length = this.tracerObj["arrow"+i.toString()].length;
```

```

      //Determines the starting position of the arrow
      let position1 = [0,0];
      //The starting position is equal to where the previous arrow pointed to
      //An exception is made for the first arrow
      if(i!=0){
        position1 = this.tracerObj["arrow"+(i-1).toString()].pointingTo.slice();
      }

      //The position the arrow points to is calculated based on angle and length
      let position2 = [0,0];
      position2[0] = Math.cos(angle)*length+position1[0];
      position2[1] = Math.sin(angle)*length+position1[1];
      //The position the arrow points to is stored in the arrow object
      this.tracerObj["arrow"+i.toString()].pointingTo = position2.slice();

      //The arrow is drawn unless it is the first
      if(i!=0){
        draw.drawArrow(ctxArrows,position1,position2,"#FCBE40");
        this.currentVal = position2;
      } else if(i==0){
        //Adds the origin
        ctxArrows.fillStyle = "#FCBE40";
        ctxArrows.beginPath();
        ctxArrows.arc(position2[0], position2[1], 3, 0, 2 * Math.PI);
        ctxArrows.fill();
      }
    }
    this.updateTrail();
  }



  //Logs all values that a IDFT have the corresponding coefficients will run thorugh
  printValues(coefficients, delta){
    let result_string = "";
    for(let frame = 0; frame < coefficients.length; frame += delta){
      let temp = calc.c_idft(coefficients, frame);
      result_string+=temp[0].toString()+" "+(-temp[1]).toString()+"\n";
    }
    let temp = calc.c_idft(coefficients, 0);
    result_string+=temp[0].toString()+" "+(-temp[1]).toString()+"\n";
    console.log(result_string);
  }

  //Creates a trail behind the last arrow
  updateTrail(){
    this.trailLog.unshift(this.currentVal)
    if(this.Frame > 2*Math.PI - 0.5){
      this.trailLog.pop()
    }

    for(let i = 0; i < this.trailLog.length-1; i++){
      draw.drawLine(ctxArrows,this.trailLog[i],this.trailLog[i+1],"#FCBE40");
    }
  }
}
```

Listing 6: draw.js

```
//This file contains all functions related drawing preset shapes

import * as calc from "./calculation.js"

//Draws an arrow
export function drawArrow(context, position1, position2, color){

  //Draws shaft
  drawLine(context, position1, position2, color);


  //Draw arrowhead
  const trianglePath = new Path2D();
  let distance = [position2[0]-position1[0],position2[1]-position1[1]];

  //Determining size of head based on arrow length
  let headSize = calc.mgn(distance)/3;
  headSize = Math.max(Math.min(headSize,15),4);

  trianglePath.moveTo(position2[0],position2[1]);

  //Determine angle of head to line
  let angle = Math.atan(distance[1]/distance[0]);
  if(distance[0]<0){
    angle += Math.PI;
  }

  //Moves anti-clockwise
  //Side 1
  let side1 = [0,0];
```

```
  side1[0] = Math.cos(Math.PI*5/6+angle)*headSize;
  side1[1] = Math.sin(Math.PI*5/6+angle)*headSize;
  trianglePath.lineTo(position2[0]+side1[0],position2[1]+side1[1]);
  //Side 2
  let side2 = [0,0];
  side2[0] = Math.cos(Math.PI*7/6+angle)*headSize;
  side2[1] = Math.sin(Math.PI*7/6+angle)*headSize;
  trianglePath.lineTo(position2[0]+side2[0],position2[1]+side2[1]);
  //Fill shape
  context.fillStyle = color;
  context.fill(trianglePath);

}

//Draws a line according to the given values
export function drawLine(context, position1, position2, color){
  const line = new Path2D();

  line.moveTo(position1[0],position1[1]);
  line.lineTo(position2[0],position2[1]);

  context.strokeStyle = color;
  context.stroke(line);
}

//Draws a cross according to the given values
export function drawCross(context, position, size, color){
  const cross = new Path2D();

  cross.moveTo(position[0] + size/2, position[1] + size/2);
  cross.lineTo(position[0] - size/2, position[1] - size/2);
  cross.moveTo(position[0] + size/2, position[1] - size/2);
  cross.lineTo(position[0] - size/2, position[1] + size/2);

  context.strokeStyle = color;
  context.stroke(cross);
}

//Draws a range of crosses according to the given values
export function drawCrosses(context, positions, size, color){
  for(let i = 0; i < positions.length; i++){
    drawCross(context, positions[i], size, color);
  }
}
```

Listing 7: UI.js

```
//This file contains all functions that can modify the UI

//SVGs of play- and pause-symbols
const pathPause0 = "M0 0L35 17.5L0 35V0Z"
const pathPause1 = "M0 17.5H35L0 35V17.5Z"
const pathPlay0 = "M0 0H15V35H0V0Z"
const pathPlay1 = "M20 0H35V35H20V0Z"

//Elements
const drawerExamplesDiv = document.querySelector("#drawer_examples");
const arrowInput = document.querySelector("#arrow_number_input");
const buttonMain = document.querySelector("#button_main");
const buttonConfirm = document.querySelector("#button_confirm");
const examplesHeader = document.querySelector("#header_wrapper")
const examplesWrapper = document.querySelector("#examples_wrapper");
const svgPlayPause = document.querySelector("#play-pause");


//Load examples into example drawer
export function setProperties(){
  examplesHeader.style.height = "18px";
  drawerExamplesDiv.style.padding = "5px";

  let vh = Math.max(document.documentElement.clientHeight, window.innerHeight || 0);
  let examples = examplesWrapper.getElementsByTagName('div');
  for(let i = 0; i < examples.length; i++){
    examples[i].style.height = (0.034 * vh).toString() + "px";
    examples[i].style.marginTop = "5px";
  }
}

//Clears a selected canvas
export function resetCanvas(context,canvas){
  context.clearRect(0,0,canvas.width,canvas.height);
}

//Calculates the example drawer's height from the number of examples
function getDrawerHeight(){
  let examplesList =  examplesWrapper.getElementsByTagName('div');

  let exampleNumber = examplesList.length;
  let exampleHeight = parseFloat(examplesList[0].style.height);
  let exampleMargin = parseFloat(examplesList[0].style.marginTop);
```

```
  let headerHeight = parseFloat(examplesHeader.style.height);
  let padding = parseFloat(drawerExamplesDiv.style.padding);

  let drawerHeight = (exampleHeight + exampleMargin) * exampleNumber + 2*padding + headerHeight + 10;
  return drawerHeight;
}

//Animation that appears when opening the examples drawer
export function openDrawer(){
  drawerExamplesDiv.dataset.toggled = "true";

  //Expands the drawer upwards
  let openDrawerAnim = anime({
    duration: 200,
    easing: "easeOutExpo",
    targets: ["#drawer_examples"],
    height: getDrawerHeight(),
    autoplay: false
  })
  openDrawerAnim.play();

  //Turns around the arrow that is used to toggle the drawer
  let forwardSpinArrowAnim = anime({
    duration: 200,
    easing: "easeOutExpo",
    targets: ["#arrow"],
    rotate: 180,
    autoplay: false
  })
  forwardSpinArrowAnim.play();
}

//Animation that appears when closing the examples drawer
export function closeDrawer(){
  drawerExamplesDiv.dataset.toggled = "false";

  //Shrinks drawer to initial height
  let closeDrawerAnim = anime({
    duration: 200,
    easing: "easeOutExpo",
    targets: ["#drawer_examples"],
    height: [getDrawerHeight(),35],
    autoplay: false
  })
  closeDrawerAnim.play();

  //Turns around the arrow that is used to toggle the drawer
  let backSpinArrowAnim = anime({
    duration: 200,
    easing: "easeOutExpo",
    targets: ["#arrow"],
    rotate: 0,
    autoplay: false
  })
  backSpinArrowAnim.play();
}

//Describes animations that are initiated through the button at the bottom center
export function morphButtonMain(state){
  arrowInput.style.display = "inline-block";

  const timeline = anime.timeline({
    duration: 400,
    easing: "easeOutExpo"
  });

  //Animation that connects the drawing and customization phases
  if(state==0){
    buttonMain.style.cursor = "pointer";
    timeline.add({
      targets: ["#play-pause","#button_restart"],
      opacity: 0
    })
    timeline.add({
      targets: ["#button_main"],
      width: 280,
      translateX: -140,
      translateY: [30,30]
    }).finished;
    timeline.add({
      targets: ["#drawer_examples"],
      translateX: -105,
      translateY: 0,
      opacity: 1
    });
    timeline.add({
      targets: ["#calculation_text"],
      opacity: 1
    });
    timeline.add({
      targets: ["#button_reset"],
      opacity: 1
```

```
    });
    timeline.add({
      targets: ["#button_reset"],
      translateX: -30,
      translateY: 30
    });

  }

  //Animation that connects the customization and viewing phases
  if(state==1){
    arrowInput.value = 0;
    buttonConfirm.style.zIndex = 5;
    buttonMain.style.cursor = "default";

    timeline.add({
      targets: ["#drawer_examples"],
      translateX: [-105,-105],
      translateY: [0,30],
      opacity: [1,0]
    });
    timeline.add({
      targets: ["#calculation_text"],
      opacity: [1,0]
    });
    timeline.add({
      targets: ["#button_reset"],
      translateX: [-30,-110],
      translateY: [30,30]
    });
    timeline.add({
      targets: ["#button_reset"],
      opacity: [1,0]
    });
    timeline.add({
      targets: ["#button_main"],
      width: 130,
      translateX: [-140,-65],
      translateY: [30,30]
    }).finished;
    timeline.add({
      targets: ["#button_main"],
      translateY: [30,-10]
    })
    timeline.add({
      targets: ["#button_confirm"],
      begin: function(){
        buttonConfirm.style.display = "inline-block";
      }
    })
    timeline.add({
      targets: ["#button_confirm"],
      translateY: 15,
      translateX: -45
    })
    timeline.add({
      targets: ["#arrow_number_input","#button_confirm"],
      opacity: [0,1]
    })
  }

  //Returns main button to the initial state
  else if(state == 2){

    buttonConfirm.style.zIndex = 0;
    buttonMain.style.cursor = "pointer";

    timeline.add({
      targets: ["#button_confirm"],
      translateX: [-45,-45],
      translateY: [15,-40]
    })
    timeline.add({
      targets: ["#button_confirm","#arrow_number_input"],
      opacity: [1,0]
    })
    timeline.add({
      targets: ["#button_confirm","#arrow_number_input"],
      begin: function(){
        buttonConfirm.style.display = "none";
        arrowInput.style.display = "none";
      }
    })
    timeline.add({
      targets: ["#button_main"],
      translateY: [-10,30]
    })
    timeline.add({
      targets: ["#button_main"],
      translateX: -30,
      width: 60
    })
```

```
    timeline.add({
      targets: ["#play-pause"],
      begin: function(){
        svgPlayPause.style.display = "inline-block";
      }
    })
    timeline.add({
      targets: ["#play-pause","#button_restart"],
      opacity: 1
    })
    timeline.add({
      targets: ["#button_restart"],
      translateX: [-25,50],
      translateY: [22.5,22.5]
    })

  }
}

//Swaps the button between the play- and pause-symbols
export function togglePlayPause(state){
  //Switches to pause-symbol
  if(state == 0){
    let morphPause0 = anime({
      duration: 0,
      easing: "easeOutExpo",
      targets: ["#pp_path0"],
      d: [
        {value: pathPause0}
      ]
    })
    let morphPause1 = anime({
      duration: 0,
      easing: "easeOutExpo",
      targets: ["#pp_path1"],
      d: [
        {value: pathPause1}
      ]
    })
    let changeX = anime({
      duration: 0,
      easing: "easeOutExpo",
      targets: ["#play-pause"],
      right: 10.5
    })

    morphPause0.play();
    morphPause1.play();
    changeX.play();

  }
  //Switches to pause-symbol
  else if(state == 1){
    let morphPlay0 = anime({
      duration: 0,
      easing: "easeOutExpo",
      targets: ["#pp_path0"],
      d: [
        {value: pathPlay0}
      ]
    })
    let morphPlay1 = anime({
      duration: 0,
      easing: "easeOutExpo",
      targets: ["#pp_path1"],
      d: [
        {value: pathPlay1}
      ]
    })
    let shiftX = (parseFloat(buttonMain.style.width) -  svgPlayPause.width.animVal.value)/2;
    let changeX = anime({
      duration: 0,
      easing: "easeOutExpo",
      targets: ["#play-pause"],
      right: shiftX
    })

    morphPlay0.play();
    morphPlay1.play();
    changeX.play();
  }
}
```

The following two documents hold the coordinate values of two examples:

Listing 8: KSimLee.txt

```
-232, -81
-285, -54
```

```
-268, -54
-268, -35
-464, -35
-464, -26
-453, -26
-453, 46
-489, 46
-473, 71
-408, 71
-401, 81
401, 81
408, 71
473, 71
489, 46
453, 46
453, -26
464, -26
464, -35
268, -35
268, -54
285, -54
232, -81
```

Listing 9: PI.txt

```
-118, -45
-109, -45
-92, -69.5
-75.5, -77.5
-45.5, -77.5
-48.5, -31
-62, 21.5
-77.5, 50
-99, 82
-96, 100
-80, 112.5
-57.5, 109
-38.5, 70.5
-31.5, 21.5
-27.5, -21
-23, -77.5
28.5, -77.5
25, -34.5
19.5, 38.5
19.5, 80
36.5, 106
73.5, 112
99, 97
113, 72
116, 46.5
108.5, 46.5
99, 69
79, 75.5
58, 62
53, 18.5
58, -26.5
60.5, -76.5
116.5, -76.5
116.5, -112.5
-35, -112.5
-64.5, -110
-88, -102
-101, -87
```

Appendix C Source Code Visualization DQFT

This section contains the code that describes the program that was used to visualize the Discrete Quaternion Fourier Transform. It has been split into three documents.

Listing 10: index.html

```html
<!DOCTYPE html>
<html lang="en">

<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <meta http-equiv="X-UA-Compatible" content="ie=edge">
  <title>Quaternion Fourier Transform</title>

  <!-- CSS file -->
  <link href="styles.css" rel="stylesheet">
  <!-- script -->
  <script src="script.js" defer></script>
</head>

<body>
  <div id="wrapper">
    <!-- allows to toggle whether the 4th dimension is shown -->
    <div id="div_checkbox">
      <input type="checkbox" id="check_display" checked>
      </input>
      Display 4th Dimension
    </div>

    <!-- can be used to turn the view -->
    <input type="range" min="0" max="6.2830" value="0" id="slider" step="0.01">

    <canvas id="canvas"></canvas>

    <!-- input menu on the right -->
    <div id="input_rec">
      <div id="add_point">
        <div class="point_wrapper">
          <div class="plus_wrapper">
            <div class="plus" id="plus"></div>
            <div class="hitbox" id="hitbox"></div>
          </div>
          <div id="input_wrapper">
            <input type="number" class="coordinate" id="xValIn"></input>
            <input type="number" class="coordinate" id="yValIn"></input>
            <input type="number" class="coordinate" id="zValIn"></input>
          </div>
        </div>
      </div>
    </div>

  </div>
</body>

</html>
```

Listing 11: styles.css

```css
* {
  /* appearance */
  padding: 0;
  margin: 0;
}

html, body {
  /* appearance */
  height: 100vh;
  margin: 0;
  background-color: #32373e;
  overflow: hidden;
}

#wrapper {
  /* apperance */
  height: 100%;
  width: 100%;
}

#input_rec {
  /* apperance */
  --height: 80%;
  height: var(--height);
  width: 6.5cm;
  border-radius: 20px;
```

```
  background-color: #5b5d60;

  /* position */
  position: fixed;
  right: 3%;
  top: calc(50% - calc(var(--height) / 2));
  z-index: 10;

  /* children */
  align-items: center;
}

.point{
  /* apperance */
  width: 90%;
  height: 10%;
  background-color: #73767C;
  margin: auto;
  margin-top: 5%;
  border-radius: 10px;

  /* position */
  position: relative;

  /* children */
  text-align: center;
}

#add_point{
  /* appearance */
  width: 90%;
  height: 10%;
  background-color: #73767C;
  border-radius: 10px;

  /* position */
  position: absolute;
  left: 5%;
  bottom: 1.5%;

  /* children */
  text-align: center;
}

.point_wrapper{
  /* appearance */
  height: 50%;
  width: 100%;

  /* position */
  position: relative;
  top: 25%;

  /* children */
  text-align: center;
}

.coordinate{
  /* appearance */
  type: number;
  height: 100%;
  width: 20%;
  margin: 3px;
  margin-top: 0;
  background-color: #CBCDD1;
  border: 0;
  border-radius: 4px;

  /* children */
  text-align: center;
}

.coordinate:focus{
  /* apperance */
  outline: none;
  outline-color: transparent;
  border: solid black;
  border-width: 2px;
  margin-top: -4px;
  margin-right:1px;
  margin-left:1px;
}

.arrows{
  /* appearance */
  width: 16%;
  height: 100%;

  /* position */
  position: absolute;

  /* children */
```

```
  text-align: center;
}

.arrow{
  /* appearance */
  border: solid black;
  border-width: 0 3px 3px 0;
  margin: auto;
  margin-left:-4px;
  padding: 3px;
  opacity: 0.3;

  /* position */
  position: absolute;
  left: 50%;

  /* misc */
  cursor: pointer;
}

.upArrow{
  /* appearance */
  border: solid black;
  border-width: 0 3px 3px 0;
  margin: auto;
  margin-left:-4px;
  padding: 3px;
  opacity: 0.3;

  /* position */
  position: absolute;
  left: 50%;
  top: 15%;
  transform: rotate(-135deg);

  /* misc */
  cursor: pointer;
}

.downArrow{
  /* appearance */
  border: solid black;
  border-width: 0 3px 3px 0;
  margin: auto;
  margin-left:-4px;
  padding: 3px;
  opacity: 0.3;

  /* position */
  position: absolute;
  left: 50%;
  bottom: 15%;
  transform: rotate(45deg);

  /* misc */
  cursor: pointer;
}

.downArrow:hover, .upArrow:hover {
  /* appearance */
  opacity: 1;
}

.coordinate::-webkit-outer-spin-button,
.coordinate::-webkit-inner-spin-button {
  /* appearance */
  -webkit-appearance: none;
  margin: 0;
}

.cross {
  /* appearance */
  width: 100%;
  height: 100%;
  margin-top: 10%;
  margin-left: -3px;
  opacity: 0.3;

  /* position */
  position: absolute;

  /* misc */
  cursor: pointer;
}
.cross:hover {
  /* appearance */
  opacity: 1;
}
.cross:before, .cross:after {
  /* appearance */
  position: absolute;
  content: ' ';
```

```
  height: 80%;
  width: 3px;
  background-color: #000000;
}
.cross:before {
  /* position */
  transform: rotate(45deg);
}
.cross:after {
  /* position */
  transform: rotate(-45deg);
}

.plus_wrapper:hover .plus{
  /* appearance */
  opacity: 1;
}

.plus {
  /* apperance */
  width: 100%;
  height: 100%;
  opacity: 0.3;
  margin-top: 4%;

  /* position */
  position: absolute;
}

.plus:before, .plus:after {
  /* appearance */
  width: 3px;
  height: 80%;
  content: ' ';
  background-color: #000000;

  /* position */
  position: absolute;
}
.plus:after {
  /* position */
  transform: rotate(-90deg);
}

.hitbox {
  /* appearance */
  width:100%;
  height:100%;

  /* position */
  position: absolute;
  z-index: 10;

  /* misc */
  cursor: pointer;
}


.cross_wrapper{
  /* appearance */
  width: 16%;
  height: 70%;

  /* position */
  position: absolute;
  right: 0%;
  top: 15%;

  /* children */
  text-align: center;
}

.plus_wrapper{
  /* appearance */
  width: 22.5%;
  height: 70%;

  /* position */
  position: absolute;
  right: 0%;
  top: 15%;

  /* children */
  text-align: center;
}

#input_wrapper {
  /* appearance */
  height: 100%;
  width: 100%;

  /* position */
```

```
  position: relative;
  left: -7%;
}

#div_checkbox {
  /* apperance */
  display:flex;
  height: 4vh;
  border-radius: 10px;
  padding-left: 1.5vh;
  padding-right: 1.5vh;
  margin-top:1vh;
  margin-bottom: 1vh;
  background-color: #5b5d60;
  color: #CBCDD1;

  /* position */
  position: fixed;
  left: 50%;
  bottom: 10%;
  transform: translate(-50%,0);

  /* font */
  vertical-align: middle;
  line-height: 4vh;
  font-size:2vh;
  font-family: Helvetica;
  align-items: center;
  justify-content: center;
}

#check_display {
  /* appearance */
  width: 2vh;
  height: 100%;
  margin-right: 1vh;

  /* position */
  position: relative;
}

#check_display:checked {
  /* appearance */
  color: #FCBE40;
  background-color: #FCBE40;
}

#label_check {
  /* appearance */
  height: 100%;
  margin-left: 6px;

  /* position */
  position: relative;
  top: 50%;
  transform: translateY(-1vh);

  /* font */
  font-size: 2vh;
}

#slider {
  /* appearance */
  --width: 30%;
  width: var(--width);
  overflow: hidden;
  -webkit-appearance: none;
  background-color: #5b5d60;

  /* position */
  position: fixed;
  left: calc(50% - calc(var(--width) / 2));
  bottom: 17%;
}

#slider::-webkit-slider-runnable-track {
  /* appearance */
  margin-top: -1px;
  -webkit-appearance: none;
  color: #FCBE40;
}

#slider::-webkit-slider-thumb {
  /* appearance */
  width: 20px;
  height: 10px;
  -webkit-appearance: none;
  background: #FCBE40;

  /* misc */
  cursor: ew-resize;
}
```

Listing 12: script.js

```
//Canvases
const canvas = document.querySelector("#canvas");
const ctx = canvas.getContext('2d');
const canvasList = ["#canvas"]


//UI
const slider = document.querySelector("#slider");
const pointDisplay = document.querySelector("#input_rec");
const hitbox = document.querySelector("#hitbox");
const inputX = document.querySelector("#xValIn");
const inputY = document.querySelector("#yValIn");
const inputZ = document.querySelector("#zValIn");
const checkboxDisplay = document.querySelector("#check_display");


//Control how the three-dimensional space is displayed
let origin = [0,0]
const scale = 14
let rotation = 0
let transform = [[1,0,0],[0,1,0],[0,0,1]]

//Stores current frame
let frame = 0;

//Store current points
let set = randomSet(5,4,20);
let pointCounter = 0;
//Combines set and pointCounter
let pointList = [];

//Equals the mu used in the DQFT
let u = [0,0,0,1];




//* UI *//


window.addEventListener("resize",resizeWindow);


//Prevents refreshing through pulling down on Safari
if (window.safari) {
  history.pushState(null, null, location.href);
  window.onpopstate = function() {
      history.go(1);
  };
}


//Updates the view whenever it is rotated
slider.oninput = function(){
  update();
};


//Adds funcitonality to the plus button that allows points to be added
hitbox.addEventListener('click', () => {
  //Ensure suitable values have been chosen
  if(inputX.value && inputY.value && inputZ.value && inputX.value >= 0 && inputY.value >= 0 && inputZ.value >= 0){
    if(inputX.value <= 20 && inputY.value <= 20 && inputZ.value <= 20){

      if(pointCounter==0){
        set = [];
      }
      pointCounter++;

      //Selects a random fourth dimension
      let r = Math.floor(Math.random()*20)

      pointList.push([pointCounter,[r,inputX.value,inputY.value,inputZ.value]]);
      updatePointDisplay();
    }
  }
});


//Adds a point of certain values to the list along with its HTML element
function addPoint(x,y,z,name){
  let input = [x,y,z]

  //Describes HTML elements
  let instructions = {
    point: ["#input_rec","div","point","point"],
    point_wrapper: ["#point","div","point_wrapper","point_wrapper"],
    arrows: ["#point_wrapper","div","arrows","arrows"],
    upArrow: ["#arrows","div","upArrow","upArrow"],
    downArrow: ["#arrows","div","downArrow","downArrow"],
```

```
    cross_wrapper: ["#point_wrapper","div","cross_wrapper","cross_wrapper"],
    cross: ["#cross_wrapper","div","cross","cross"],
    xVal: ["#point_wrapper","input","coordinate","xVal"],
    yVal: ["#point_wrapper","input","coordinate","yVal"],
    zVal: ["#point_wrapper","input","coordinate","zVal"],
  }

  //Adjusts various properties
  let propArr = Object.keys(instructions);
  for(let i = 0; i < propArr.length; i++){
    instructions[propArr[i]][3]+="_"+name;
    if(i!=0){
      instructions[propArr[i]][0]+="_"+name;
    }
  }

  //Contrsucts HTML Elements
  for(let i = 0; i < propArr.length; i++){
    let node = document.createElement(instructions[propArr[i]][1])
    node.setAttribute("class",instructions[propArr[i]][2]);
    node.setAttribute("id",instructions[propArr[i]][3]);

    //Adds individual properties

    if(instructions[propArr[i]][2]=="coordinate"){
      node.setAttribute("type","number");
      node.setAttribute("value",input[i-7]);
    }

    if(instructions[propArr[i]][2]=="upArrow"){
      node.addEventListener('click', () =>{
        let pointInfo = pointList.find(element => element[0]==name);
        let pointIndex = pointList.indexOf(pointInfo);
        if(pointIndex>0){
          let temp = pointList[pointIndex-1];
          pointList[pointIndex-1]= pointInfo;
          pointList[pointIndex] = temp;
          updatePointDisplay();
        }
      })
    }

    if(instructions[propArr[i]][2]=="cross"){
      node.addEventListener('click', () =>{
        document.querySelector("#point"+"_"+name).remove();
        let pointInfo = pointList.find(element => element[0]==name);
        let pointIndex = pointList.indexOf(pointInfo);
        pointList.splice(pointIndex,1);
        updatePointDisplay();
      })
    }

    document.querySelector(instructions[propArr[i]][0]).appendChild(node);
  }
}


//Refreshes point menu on the right to match current points
function updatePointDisplay(){
  console.log(pointList);

  let childCount = pointDisplay.children.length;
  for(let i = 1; i < childCount; i++){
    pointDisplay.children[1].remove();
  }

  set = [];

  for(let i = 0; i<pointList.length; i++){
    addPoint(pointList[i][1][1],pointList[i][1][2],pointList[i][1][3],pointList[i][0]);
    set[i]=[pointList[i][1][0],pointList[i][1][1],pointList[i][1][2],pointList[i][1][3]];
  }

  frame = 0;
}




// ANIMATION


//Animation is started upon opening the program
runAnimation();



let arrowAnim;
//Repeats the update function
function runAnimation(){
  setTimeout(function(){
    update();
```

```
    frame += 0.003;
    arrowAnim = window.requestAnimationFrame(function(){runAnimation()});
  }, 10);
}


//Updates the three-dimensional space
function update(){
  rotation = slider.value;
  ctx.clearRect(0,0,canvas.width,canvas.height);

  //Axis
  arrow3D(ctx,[0,0,0],[20,0,0],"#BAB7AC");
  arrow3D(ctx,[0,0,0],[0,20,0],"#BAB7AC");
  arrow3D(ctx,[0,0,0],[0,0,20],"#BAB7AC");

  //Draws the arrows that make up the IDQFT
  if(set.length > 1){
    for(let i=1;i<IDQFT(set.length-1+frame,DQFT(set),true).length;i++){
      arrow3D(ctx,[IDQFT(set.length-1+frame,DQFT(set),true)[i-1][1],IDQFT(set.length-1+frame,DQFT(set),true)[i-1][2],IDQFT(
            set.length-1+frame,DQFT(set),true)[i-1][3]],[IDQFT(set.length-1+frame,DQFT(set),true)[i][1],IDQFT(set.length-1+frame,
            DQFT(set),true)[i][2],IDQFT(set.length-1+frame,DQFT(set),true)[i][3]],"#FCBE40");
    }
  }

  //Draws trail
  for(let i=frame;i<=set.length-1+frame;i+=0.01){
    line3D([IDQFT(i,DQFT(set))[1],IDQFT(i,DQFT(set))[2],IDQFT(i,DQFT(set))[3]],[IDQFT(i+0.01,DQFT(set))[1],IDQFT(i+0.01,DQFT(set)
          )[2],IDQFT(i+0.01,DQFT(set))[3]],getColor(IDQFT(i,DQFT(set))[0]));
  }

  //Drwas the various points
  for(let i=0;i<set.length;i+=1){
    cross3D(ctx,[set[i][1],set[i][2],set[i][3]],7,getColor(set[i][0]));
  }
}




// DRAWING


//Will draw a cross of given properties on a two-dimensional plane
function drawCross(context, position, size, color){
  const cross = new Path2D();

  cross.moveTo(position[0] + size/2, position[1] + size/2);
  cross.lineTo(position[0] - size/2, position[1] - size/2);
  cross.moveTo(position[0] + size/2, position[1] - size/2);
  cross.lineTo(position[0] - size/2, position[1] + size/2);
  context.strokeStyle = color;
  context.stroke(cross);
}


//Will draw a cross of given properties in a three-dimensional space
function cross3D(context, p1, size, color){
  let position=render3D(p1);
  const cross = new Path2D();

  cross.moveTo(position[0] + size/2, position[1] + size/2);
  cross.lineTo(position[0] - size/2, position[1] - size/2);
  cross.moveTo(position[0] + size/2, position[1] - size/2);
  cross.lineTo(position[0] - size/2, position[1] + size/2);
  context.strokeStyle = color;
  context.stroke(cross);
}



function dot3D(context, p1, size, color){
  let position=render3D(p1);
  const dot = new Path2D();

  dot.arc(position[0],position[1],size,0,2*Math.PI);
  context.fillStyle = color;
  context.fill(dot);
}



function line3D(p1,p2,color="#FCBE40"){
  const line = new Path2D();
  line.moveTo(render3D(p1)[0],render3D(p1)[1]);
  line.lineTo(render3D(p2)[0],render3D(p2)[1]);
  ctx.strokeStyle = color;
  ctx.stroke(line);
}



```

```
function arrow3D(context, p1, p2, color){

  let position1 = render3D(p1);
  let position2 = render3D(p2);

  //Draw line
  const line = new Path2D();

  line.moveTo(position1[0],position1[1]);
  line.lineTo(position2[0],position2[1]);
  context.strokeStyle = color;
  context.stroke(line);


  //Draw arrowhead
  const trianglePath = new Path2D();
  let distance = [position2[0]-position1[0],position2[1]-position1[1]];

  //Determining size of head based on arrow length
  let headSize = mgn([p1[0]-p2[0],p1[1]-p2[1],p1[2]-p2[2]]);
  headSize = Math.max(Math.min(headSize,15),4);

  trianglePath.moveTo(position2[0],position2[1]);

  //Determine angle of head to line
  let angle = Math.atan(distance[1]/distance[0]);
  if(distance[0]<0){
    angle += Math.PI;
  }

  //Moves anti-clockwise
  //Side 1
  let side1 = [0,0];
  side1[0] = Math.cos(Math.PI*5/6+angle)*headSize;
  side1[1] = Math.sin(Math.PI*5/6+angle)*headSize;
  trianglePath.lineTo(position2[0]+side1[0],position2[1]+side1[1]);
  //Side 2
  let side2 = [0,0];
  side2[0] = Math.cos(Math.PI*7/6+angle)*headSize;
  side2[1] = Math.sin(Math.PI*7/6+angle)*headSize;
  trianglePath.lineTo(position2[0]+side2[0],position2[1]+side2[1]);
  //Fill shape
  context.fillStyle = color;
  context.fill(trianglePath);

}




// CALCULATION


//The Discrete Quaternion Fourier Transform
function DQFT(values){
  let result = [];
  let M = values.length;

  for(let t=0;t<=M-1;t++){
    let subtotal = [0,0,0,0]
    for(let x=0;x<=M-1;x++){
      let summand = q_mult(values[x],q_exp(q_mult(u,[-2*Math.PI*(x*t/M),0,0,0])));
      subtotal = q_add(subtotal, summand);
    }
    result.push([t,q_mult([1/Math.pow(M,0.5),0,0,0],subtotal)]);
  }

  return result;
}


//The Inverse Discrete Quaternion Fourier Transform
function IDQFT(t,values,subs=false){
  let subtotals = [];
  let total = [0,0,0,0];
  let M = values.length;

  for(let x=0;x<=M-1;x++){
    let summand = q_mult(values[x][1],q_exp(q_mult(u,[2*Math.PI*(values[x][0]*t/M),0,0,0])));
    total = q_add(total, q_mult([1/Math.pow(M,0.5),0,0,0],summand));
    subtotals.push(total);
  }


  if(subs==true){
    return subtotals;
  } else {
    return total;
  }
}

```

```
//Will calculate the length of a vector
function mgn(vec){
  let result = 0
  for(let i=0;i<vec.length;i++){
    result += Math.pow(vec[i],2);
  }
  result = Math.pow(result,0.5);
  return result
}


//Extracts the vector part of a quaternion
function Vec(quaternion){
  return [quaternion[1],quaternion[2],quaternion[3]]
}


//The exponential function for quaternions
function q_exp(q){
  let mgnSc = mgn(Vec(q))
  let result=[0,0,0,0]

  result[0]=Math.pow(Math.e,q[0])*Math.cos(mgnSc)

  if(mgnSc!=0){
    for(let i=1; i<4; i++){
      result[i] =  Math.pow(Math.e,q[0])*(q[i]/mgnSc)*Math.sin(mgnSc)
    }
  }
  return result
}


function q_add(p,q){
  return [p[0]+q[0],p[1]+q[1],p[2]+q[2],p[3]+q[3]];
}


function q_sub(p,q){
  return [p[0]-q[0],p[1]-q[1],p[2]-q[2],p[3]-q[3]];
}


function q_mult(p,q){
  let result = [0,0,0,0];

  result[0]=p[0]*q[0]-p[1]*q[1]-p[2]*q[2]-p[3]*q[3];
  result[1]=p[0]*q[1]+p[1]*q[0]-p[2]*q[3]+p[3]*q[2];
  result[2]=p[0]*q[2]+p[1]*q[3]+p[2]*q[0]-p[3]*q[1];
  result[3]=p[0]*q[3]-p[1]*q[2]+p[2]*q[1]+p[3]*q[0];

  return result;
}


//Apply a 3x3 projection to a given vector
function applyProjection(mtx,vec3){
  let result = [0,0,0];
  result[0] = mtx[0][0]*vec3[0]+mtx[0][1]*vec3[1]+mtx[0][2]*vec3[2];
  result[1] = mtx[1][0]*vec3[0]+mtx[1][1]*vec3[1]+mtx[1][2]*vec3[2];
  result[2] = mtx[2][0]*vec3[0]+mtx[2][1]*vec3[1]+mtx[2][2]*vec3[2];
  return result;
}


//Will multiply two 3x3 matrices
function mtx_mult(mtx1,mtx2){
  let result = [[0,0,0],[0,0,0],[0,0,0]];
  result[0][0] = mtx1[0][0]*mtx2[0][0]+mtx1[0][1]*mtx2[1][0]+mtx1[0][2]*mtx2[2][0];
  result[1][0] = mtx1[1][0]*mtx2[0][0]+mtx1[1][1]*mtx2[1][0]+mtx1[1][2]*mtx2[2][0];
  result[2][0] = mtx1[2][0]*mtx2[0][0]+mtx1[2][1]*mtx2[1][0]+mtx1[2][2]*mtx2[2][0];
  result[0][1] = mtx1[0][0]*mtx2[0][1]+mtx1[0][1]*mtx2[1][1]+mtx1[0][2]*mtx2[2][1];
  result[1][1] = mtx1[1][0]*mtx2[0][1]+mtx1[1][1]*mtx2[1][1]+mtx1[1][2]*mtx2[2][1];
  result[2][1] = mtx1[2][0]*mtx2[0][1]+mtx1[2][1]*mtx2[1][1]+mtx1[2][2]*mtx2[2][1];
  result[0][2] = mtx1[0][0]*mtx2[0][2]+mtx1[0][1]*mtx2[1][2]+mtx1[0][2]*mtx2[2][2];
  result[1][2] = mtx1[1][0]*mtx2[0][2]+mtx1[1][1]*mtx2[1][2]+mtx1[1][2]*mtx2[2][2];
  result[2][2] = mtx1[2][0]*mtx2[0][2]+mtx1[2][1]*mtx2[1][2]+mtx1[2][2]*mtx2[2][2];
  return result;
}




// RENDERING


//Converts a three-dimensional point to a two-dimensional point on screen
function render3D(vector3){
  let vec2 = [0,0]
  let vec3 = applyProjection(rotate3D(transform,rotation),vector3);
  vec2[0] = (-Math.pow(3,0.5)/2)*vec3[0]+(Math.pow(3,0.5)/2)*vec3[1]
  vec2[1] = -(-0.5*vec3[0]-0.5*vec3[1]+vec3[2])
```

```
  vec2[0] *= scale
  vec2[1] *= scale
  vec2[0] += origin[0]
  vec2[1] += origin[1]
  return vec2
}


//Rotates a matrix around the z-axis by a given angle
function rotate3D(mtx,angle){
  let mtx_rotation = [[Math.cos(angle),-Math.sin(angle),0],[Math.sin(angle),Math.cos(angle),0],[0,0,1]];
  return mtx_mult(mtx,mtx_rotation);
}



// MISC


//Generates a random array of n number with a maximum value of max
function randomArray(n, max){
  let arr = []
  for(let i=0;i<n;i++){
    arr.push(Math.floor(Math.random()*max));
  }
  return arr
}


//Generates a set of n arrays with size numbers and a maximum value of amx
function randomSet(n, size, max){
  let set = []
  for(let i=0;i<n;i++){
    set.push(randomArray(size,max));
  }
  return set
}


//Picks a color on a linear scale between red and blue
function getColor(val){

  if(checkboxDisplay.checked == false){
    return "#FCBE40";
  }

  let col1 = [0,218,255]
  let col2 = [176,126,26]

  let r = Math.round(val/20*(col1[0]-col2[0])+col2[0]);
  let r0 = Math.floor(r/16);
  let r1 = (r/16-r0)*16;

  let g = Math.round(val/20*(col1[1]-col2[1])+col2[1]);
  let g0 = Math.floor(g/16);
  let g1 =(g/16-g0)*16;

  let b = Math.round(val/20*(col1[2]-col2[2])+col2[2]);
  let b0 = Math.floor(b/16);
  let b1 =(b/16-b0)*16;

  return "#"+r0.toString(16)+r1.toString(16)+g0.toString(16)+g1.toString(16)+b0.toString(16)+b1.toString(16);
}


//Makes variuos adjusments when the window is resized
resizeWindow();
function resizeWindow() {
  for(let i = 0; i < canvasList.length; i++){
    let canvas = document.querySelector(canvasList[i]);
    canvas.height = window.innerHeight;
    canvas.width = window.innerWidth;
  }

  origin = [window.innerWidth/2,window.innerHeight/2]
}
```