# Logical Gates by Code Deformation in Topological Quantum Codes

*Thomas Rowan Scruby*

A dissertation submitted in partial fulfillment

of the requirements for the degree of

**Doctor of Philosophy**

of

**University College London**.

Department of Physics

University College London

September 21, 2021

I, Thomas Rowan Scruby, confirm that the work presented in this thesis is my own. Where information has been derived from other sources, I confirm that this has been indicated in the work.

# Abstract

Quantum error correcting codes (QECCs) allow us to protect qubits from noise and are expected to be essential features of any kind of scalable, fault-tolerant quantum computer. By encoding information in a QECC we make unintentional modification of that information less likely, but also make intentional modification more difficult. Operations that perform such modifications are referred to as "logical operations" or "logical gates" and a common, fault-tolerant approach to performing these operations is the use of "transversal" logical gates. However, a fundamental theorem of quantum error correction is that no QECC can possess a universal set of transversal gates.

An alternate approach to performing logical gates is the technique of code deformation, which involves a sequence of modifications (deformations) of the code which transform the encoded information. In the class of QECCs called topological codes these deformations have natural mathematical interpretations in terms of transformations of a manifold, and physical interpretations in terms of the motions of quasiparticles in certain condensed matter systems.

Here we examine two different code deformation techniques. The first is the braiding of a certain type of defect (a twist defect) in multiple copies of the two-dimensional surface code. We classify the set of logical operations which can be performed in this fashion by drawing a connection to the braiding relations of a hierarchy of anyon models. The second example involves switching between two- and three-dimensional versions of a code and an unorthodox method of decoding called just-in-time (JIT) decoding. We numerically demonstrate the existence of a threshold for this decoding strategy in surface codes and then proceed to examine the errors that occur if partial transversal gates are interleaved with this procedure.

# Impact Statement

Quantum computers promise to outperform their classical counterparts in a number of areas, including factorisation of prime numbers (and therefore breaking of RSA encryption), database searches and simulation of condensed matter and chemical systems. However, because of the high levels of noise inherent to most qubit implementations it is expected that some kind of error correction scheme will be a necessary part of any practical quantum computer. The choice of such an error correction scheme will impose restrictions on the architecture of the computer as many error correcting codes require specific qubit connectivities and geometries, and additional resources are often required in order to perform computation with the encoded information. As such, the identification of codes with low resource costs but high error tolerances is an important research direction.

The results in this thesis contribute towards an understanding of how logical operations in error correcting codes may be implemented and which implementations are most resource efficient. This may inform the planning and design of quantum computing architectures in future experimental work, and in particular the numerical results obtained for the just-in-time decoding procedure in chapter 3 suggest that further optimisation of this scheme may make it experimentally viable in the future.

Additionally, these results may be useful for future theoretical work on this topic. For instance, the investigation of Clifford errors in the three-dimensional surface code presented in chapter 4 may be relevant to future analytic and numerical work in this code, and the techniques used in that chapter may also be generalised to give similar results in other codes where this phenomenon has not yet been studied.

# Acknowledgements

Thanks go to my supervisor, Dan Browne, for providing invaluable support, insight and advice throughout my PhD. Thanks also to the current and former members of Dan's group (in particular to Mike Vasmer and Simon Burton) for many useful discussions. In the wider academic community, I would like to thank Paul Webster and Ben Brown for many valuable conversations and ideas, as well as Jiannis Pachos for the opportunity to visit his research group in Leeds.

Thanks to all the members of my CDT cohort (+ honourary member Fariha), my pre-covid officemates (Mike, Alex, Fagin, Zack and Andrea), my post-covid office/flatmates (Livia and Donald) and all my other friends in and out of London.

Thanks to my family for not asking me to explain my research to them (mostly).

# Contents

# List of Figures

# List of Tables

# Chapter 1

# Introduction

Quantum computers promise to outperform their classical counterparts in a number of areas, including factorisation of prime numbers [Shor, 1994], database searches [Grover, 1996] and simulation of condensed matter and chemical systems [O'Malley et al., 2016, Hempel et al., 2018]. Numerous hardware developments in recent years have lead to an increase in both the quantity and quality of qubits in prototype quantum devices, with recent experimental results showing modern quantum devices outperforming their classical counterparts at certain tasks [Arute et al., 2019]. However, these qubits are still too noisy to be of much practical use. Further developments may improve this, but noise rates low enough that large-scale algorithms such as those described above can be performed without error are not generally considered achievable. Instead, we expect that quantum error correcting codes will allow us to counteract the effects of noise [Shor, 1995, Steane, 1996a, Campbell et al., 2017]. These codes work by encoding the state of a single "logical" qubit in the state of multiple physical qubits such that errors on a small number of these physical qubits can be detected and corrected without damaging the encoded logical information. As long as the noise rate in the physical qubits is below a certain value, called a threshold, the logical error rate can be reduced by increasing the number of physical qubits in the code [Dennis et al., 2002], allowing us to achieve arbitrarily low logical error rates at the cost of additional hardware. A quantum computer that protects its logical information in this way is called *fault-tolerant*. Recent experimental results have demonstrated fault-tolerance in an error correcting

code encoding a single qubit [Egan et al., 2021].

However, an increased qubit cost is not the only price we must pay for fault-tolerance. By protecting our logical information from unintentional modification by noise but also make it more difficult for us to intentionally modify this information with logical operations. There exist a number of no-go theorems resticting the range of logical operations which are possible in various types of codes, and in general it is believed that no code possesses a computationally universal set of gates which can be implemented both unitarily and fault-tolerantly [Eastin and Knill, 2009, Zeng et al., 2011, Bravyi and König, 2013, Jochym-O'Connor et al., 2018, Burton and Browne, 2020, Webster et al., 2021, Webster and Bartlett, 2020]. As such, much attention has been given to finding non-unitary methods of implementing logical gates which may be combined with existing unitary ones to yield a universal gate set. The most popular proposal is known as magic state distillation and uses fault-tolerant operations to prepare a small number of high-fidelity resource states from a larger number of low-fidelity ones [Bravyi and Kitaev, 2005]. The overheads for this procedure can be very high [Fowler et al., 2012] (although a large amount of recent work has aimed at reducing this cost [Litinski, 2019, Campbell and Howard, 2017, Haah et al., 2017]) and so other methods of achieving universality have also been investigated. Among these are a number of techniques collectively referred to as "code deformations" which will be the focus of this thesis.

There is no precise definition of code deformation, but in general it refers to the idea of transforming one error correcting code into another (or a sequence of such transformations) in a way that is fault-tolerant. It is most commonly used when discussing a class of codes called topological codes, since these transformations can sometimes (but not always) resemble topology-preserving deformations of a manifold. This transformation may alter the encoded logical information directly or may allow access to new unitarily implementable gates which are available in the final code but not the initial one. Examples of code deformation include defect braiding [Fowler et al., 2012, Bombín, 2010, Brown et al., 2017, Webster and Bartlett, 2019, Scruby and Browne, 2020], lattice surgery [Horsman et al., 2012, Vuillot et al.,

2019] and dimension jumping [Bombín, 2016, Beverland et al., 2021, Brown, 2020, Scruby et al., 2021]. In this work we study two methods for implementing logical gates via code deformation techniques.

The first of these is the braiding of twist defects in topological codes. For each topological code there is a corresponding topological phase and errors affecting physical qubits of the code can be viewed as creation operators for excitations in this phase. Twist defects are the end-points of lattice dislocations and are associated with symmetries of the anyon model of the topological phase. In some circumstances they can reproduce the exchange statistics of non-Abelian anyons and allow us to perform gates by braiding. In chapter 2 we examine the logical gates which can be implemented by braiding twists in many copies of the most commonly studied topological code: the surface code.

The second method we will discuss is based on the process of dimension jumping, which provides a way of switching between two- and three-dimensional versions of a topological code [Bombín, 2016]. The unitary gates which can be implemented in a given topological code depend in part on the dimension of that code [Bravyi and König, 2013] and while neither the 2D or 3D code can have a universal unitary gate set individually, the combination of their gate sets can be universal. In this case arbitrary computations can be performed by using dimension jumping to switch between the two codes but this requires significantly more physical qubits than using only two-dimensional codes. In chapter 3 we discuss an example of a decoding strategy called just-in-time (JIT) decoding [Bombín, 2018a, Brown, 2020] which allows us to exchange a spatial dimension for a temporal one so that this increased qubit cost is exchanged for an increased runtime. We give an explicit description of this procedure and provide numerical evidence of a threshold for this decoding strategy. JIT decoding does not correct all errors that can occur in the code and so a threshold for JIT decoding is necessary but not sufficient to guarantee a threshold for the entire procedure. In chapter 4 we examine the effects of Clifford errors in the 3D surface code. These errors can occur when uncorrected Pauli errors (e.g. due to mistakes in the JIT decoding process) are mapped to Clifford operators

by the code's non-Clifford gate. These types of errors have previously been studied in the colour code, and we generalise these results to the surface code and show that certain features of these errors are much more easily understood in this setting.

In the remainder of this chapter we will provide an overview of the techniques that are used to model errors affecting qubits and the basic formalisms of quantum error correction. We will then introduce topological codes (with special attention given to the surface code) and finally we will discuss in more detail the various types of logical operations summarised above.

## 1.1 Qubits and Errors

A qubit is a two-state quantum system, with these states usually written $|0\rangle$ and $|1\rangle$. A general state of a qubit can be written

$$|\psi\rangle = \alpha|0\rangle + \beta|1\rangle \tag{1.1}$$

and corresponds to a normalised vector in a two-dimensional Hilbert space spanned by basis $\{|0\rangle, |1\rangle\}$. If we do not know the exact state of a qubit then this uncertainty can be represented using a density matrix $\rho = \sum_i p_i |\psi_i\rangle\langle\psi_i|$ where $|\psi_i\rangle$ are as defined in (1.1) and $p_i$ is the probability that the qubit is in state $|\psi_i\rangle$. We say that a state is pure if $\rho$ can be written as the sum of only a single $|\psi_i\rangle$, otherwise we say that it is mixed. A general density matrix state can also be written

$$\rho = \frac{I + xX + yY + zZ}{2} \tag{1.2}$$

where

$$I = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \quad X = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \quad Y = \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix} \quad Z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix} \tag{1.3}$$

are the $2 \times 2$ identity matrix and the three Pauli matrices and $x^2 + y^2 + z^2 \leq 1$. The vector $\boldsymbol{k} = (x, y, z)$ therefore defines a point in a sphere of radius 1. This sphere is

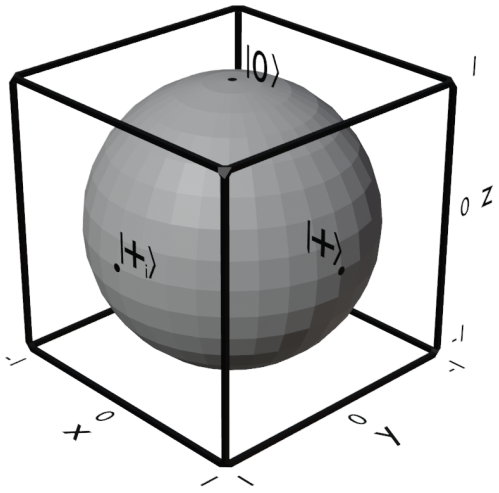



**Figure 1.1:** The Bloch sphere and the positions of the states $|0\rangle$, $|+\rangle = 1/\sqrt{2}(|0\rangle + |1\rangle)$ and $|+_i\rangle = 1/\sqrt{2}(|0\rangle + i|1\rangle)$. These three states are the $+1$ eigenstates of $Z$, $X$ and $Y$ respectively.

called the Bloch sphere (fig. 1.1) and every mixed state can be associated with a point inside of it, while every pure state can be associated with a point on its surface. The state at the centre of the sphere where $x = y = z = 0$ is called the maximally mixed state.

The state of $n$ qubits can be written as a density matrix of size $2^n \times 2^n$. If this matrix cannot be factored into a tensor product of $n$ $2 \times 2$ single-qubit density matrices then we say that the qubits are entangled.

The effects of noise on a qubit can be modelled using quantum channels. These are superoperators,

$$\mathcal{E}(\rho) = \sum_i E_i \rho E_i^\dagger, \tag{1.4}$$

where $E_i$ are called Kraus operators and satisfy $\sum_i E_i^\dagger E_i = I$. Accordingly, we say that a channel is unitary if it can be written using only a single Kraus operator. We will be particularly concerned with the following three channels: The bit-flip channel

$$\mathcal{E}_X(\rho) = (1-p)\rho + pX\rho X \tag{1.5}$$

the phase-flip channel

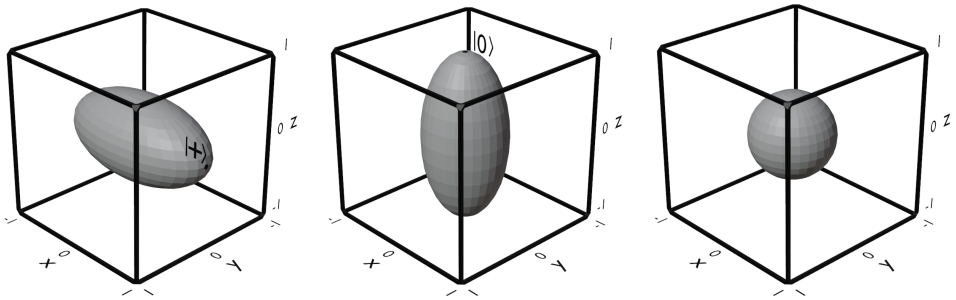



**Figure 1.2:** The action of the bit-flip, phase-flip and depolarising channels on the Bloch sphere. We can see that the state $|+\rangle$ ($|0\rangle$) is invariant under the action of the bit-flip (phase-flip) channel as this state is an eigenstate of $X$ ($Z$). Only the maximally mixed state is invariant under the action of the depolarising channel.

$$\mathcal{E}_Z(\rho) = (1-p)\rho + pZ\rho Z \tag{1.6}$$

and the depolarising channel

$$\mathcal{E}_D(\rho) = (1-p)\rho + \frac{p}{3}(X\rho X + Y\rho Y + Z\rho Z). \tag{1.7}$$

The bit-flip (phase-flip) channel applies a Pauli $X$ ($Z$) error to the qubit with probability $p$ and otherwise does nothing. The depolarising channel applies a random Pauli error to the qubit with probability $p$ and otherwise does nothing. The action of these channels can also be visualised using the Bloch sphere, as shown in fig. 1.2. The bit-flip (phase-flip) channel compresses the sphere in the $y$ and $z$ ($x$ and $y$) directions while preserving the $x$ ($z$) direction. The depolarising channel shrinks the entire sphere uniformly.

These three channels do not describe all possible errors that can affect a single qubit, and in fact there are infinitely many such errors. However, it is possible to consider only errors of this type because measurement of a Pauli operator will project general errors to Pauli errors since $X$, $Y$ and $Z$ (with complex coefficients) form a basis for the algebra of $2 \times 2$ unitary matrices. For example, consider a general rotation of the Bloch sphere about the $z$ axis by angle $\theta$. This can be written

$$\begin{pmatrix} 1 & 0 \\ 0 & e^{i\theta} \end{pmatrix} |\psi\rangle = \left( \frac{1+e^{i\theta}}{2} I + \frac{1-e^{i\theta}}{2} Z \right) |\psi\rangle . \tag{1.8}$$

$X$ and $Z$ anticommute so if we measure $X$ for this qubit we will collapse this superposition and project to either a $Z$ error or no error.

Errors which affect multiple qubits can be written as multi-qubit error channels affecting a multi-qubit density matrix. If a multi-qubit error can be written as a sequence of independent single-qubit error channels then we say that the error is uncorrelated. Correlated errors, which do not have this form, can be more difficult to deal with and will be discussed later.

## 1.2 Error Correcting Codes

### 1.2.1 Simple Codes

In classical computers errors on individual bits can be detected and corrected using error correcting codes. A very simple example is the three bit repetition code which encodes the state of a single bit in three bits. A state 000 corresponds to an encoded 0, while 111 corresponds to an encoded 1. If we measure these three bits and find they are in a state 010 then we know that either the second bit or the first and third bits have been flipped. If bit flips occur independently with probability $p$ then one error is more likely than two, so we assume that the second bit has been flipped. We can then flip the state of this bit to return to the state 000. In general we can use a majority-vote strategy to find a correction, so that if we have two 0s and a 1 we flip the 1 and vice versa. The following terminology will be used throughout the rest of this thesis

- States such as 000 and 111 which correspond to states of the encoded bit are called *logical states* or *codewords* of the code.

- An error is *detectable* if it does not map codewords to codewords. For example, any single bit flip or pair of bit flips is a detectable error in the three bit repetition code. *undetectable* errors (e.g. flipping all bits of the three bit

repetition code) are also called *logical* errors as they correspond to logical operations on the encoded bit.

- A *decoder* is an algorithm which provides a correction that maps a non-codeword to a codeword. The majority-vote strategy described above is an example of a decoder for the three bit repetition code.

- An error is *correctable* with respect to a specific decoder if the combined action of the error and the correction provided by the decoder is equivalent to the identity operation on the encoded bit. Errors can be detectable but not correctable (e.g. two bit flips in the three bit repetition code with majority-vote decoder).

- The *distance* of a code is the smallest number of single-bit operations needed to map one codeword to another (equivalently the number of bits affected by the smallest possible undetectable error). The three bit repetition code has a distance of three.

We can define a version of the three bit repetition code for qubits with logical states $\left|\overline{0}\right\rangle = |000\rangle$ and $\left|\overline{1}\right\rangle = |111\rangle$ (in general, barred states $|\overline{\psi}\rangle$ will be used to refer to logical states, while barred operators $\overline{U}$ will be used to refer to logical operators). We refer to the subspace spanned by the codewords as the *codespace*. We cannot measure the individual qubits directly to check for errors as this could damage the encoded information so instead we can make two-qubit measurements which tell us about the parity of a pair of qubits rather than the state of an individual qubit. For instance, the measurements $Z_1Z_2$ and $Z_2Z_3$ both have outcome $+1$ for $|\psi\rangle = \left|\overline{0}\right\rangle$ and $|\psi\rangle = \left|\overline{1}\right\rangle$ and so cannot be used to distinguish between these states. However, the outcome of $Z_1Z_2$ will be $-1$ if there was an $X$ error on the first or second qubit, while $Z_2Z_3$ will be $-1$ for an $X$ error on the second or third qubit. Therefore, analagously to the classical case, this code can detect up to two $X$ errors and correct a single $X$ error.

However, unlike in the classical case, bit flip errors are not the only type of error we need to protect against. A single $Z$ error on any of the three qubits maps the state

$|\overline{1}\rangle$ to $-|\overline{1}\rangle$ while acting as identity on $|\overline{0}\rangle$. In other words, a single physical $Z$ error also implements $\overline{Z}$ on the logical qubit so this code offers no protection against $Z$ errors and has distance 1. We can create a version of the repetition code which can correct $Z$ errors by instead choosing codewords $|\overline{0}\rangle = |+++\rangle$ and $|\overline{1}\rangle = |---\rangle$ (where $|+\rangle = 1/\sqrt{2}(|0\rangle + |1\rangle)$ and $|-\rangle = 1/\sqrt{2}(|0\rangle + |1\rangle)$) and making two qubit measurements $X_1X_2$ and $X_2X_3$. This allows us to detect any two-qubit $Z$ error and correct any single-qubit $Z$ error, but we can no longer detect single-qubit $X$ errors and so this code is also distance 1.

Fortunately, it is possible to find quantum error correcting codes which can protect against both $X$ and $Z$ errors. This is sufficient to protect against all errors since, as discussed above, Pauli measurements can collapse general errors to Pauli errors and $Y$ errors are the product of an $X$ and a $Z$ error (up to global phase). Perhaps the most straightforward example of this is achieved by combining the two codes discussed above (sometimes referred to as the bit-flip and phase-flip codes). We begin with the codestates of the phase-flip code

$$|\overline{0}\rangle = |+++\rangle \tag{1.9}$$

$$|\overline{1}\rangle = |---\rangle \tag{1.10}$$

and then encode each of the three qubits of this code in the bit-flip code to obtain codewords

$$|\overline{0}\rangle = \frac{1}{\sqrt{2}}(|000\rangle + |111\rangle)(|000\rangle + |111\rangle)(|000\rangle + |111\rangle) \tag{1.11}$$

$$|\overline{1}\rangle = \frac{1}{\sqrt{2}}(|000\rangle - |111\rangle)(|000\rangle - |111\rangle)(|000\rangle - |111\rangle). \tag{1.12}$$

This method of combining codes is called code concatenation and this code is called the Shor code [Shor, 1995]. It combines the $X$ and $Z$ error correcting properties of the bit-flip and phase-flip codes in order to correct arbitrary single-qubit errors. The parity measurements we make to detect errors in this code are $Z_1Z_2$, $Z_2Z_3$, $Z_4Z_5$, $Z_5Z_6$, $Z_7Z_8$, $Z_8Z_9$, $X_1X_2X_3X_4X_5X_6$ and $X_4X_5X_6X_7X_8X_9$.

### 1.2.2 Stabiliser Codes

All of the codes we have looked at so far are examples of a more general class of codes called *stabiliser* codes [Gottesman, 1997]. A stabiliser code is defined by a stabiliser group $S$, which is a group of commuting Pauli operators acting on the physical qubits of the code. The elements of $S$ are called stabilisers and have the property that

$$s|\overline{\psi}\rangle = |\overline{\psi}\rangle \quad \forall s \in S \tag{1.13}$$

for any codestate $|\overline{\psi}\rangle$. By measuring a generating set of operators of the stabiliser group we can detect any errors which anticommute with these stabilisers. The outcome of this set of measurements is called a syndrome. A generating set of stabilisers for the bit flip code is $Z_1Z_2$ and $Z_2Z_3$, while a generating set for the phase flip code is $X_1X_2$ and $X_2X_3$. Any non-stabiliser operator which commutes which all stabilisers must be a logical operator of the code because

$$U|\overline{\psi}\rangle = Us|\overline{\psi}\rangle = sU|\overline{\psi}\rangle \quad \forall s \in S \tag{1.14}$$

so $U|\overline{\psi}\rangle$ must be a codestate. General stabiliser codes can be described by the triple of integers $[\![n,k,d]\!]$ where $n$ is the number of physical qubits in the code, $k$ is the number of encoded logical qubits and $d$ is the code distance. In any stabiliser code $n-k$ is equal to the rank (number of independent generators) of the stabiliser group. For example, the bit flip and phase flip codes each have $n=3$, two stabiliser generators and each encode one qubit ($k=1$). The Shor code has $n=9$ and eight stabiliser generators so once again encodes a single qubit.

As one final piece of terminology, we define the *weight* of a stabiliser, logical operator or error to be the number of qubits of the code on which it acts nontrivially. These qubits are also referred to as the *support* of this operator. In general, it is desirable to find codes with low-weight stabilisers as this reduces requirements on qubit connectivity and can also prevent/reduce some kinds of errors which will be discussed later.

### 1.2.3 CSS Codes

An important subclass of stabiliser codes are the Calderbank-Shor-Steane (CSS) codes. These are quantum codes defined using a pair of classical codes, with $X$ stabilisers derived from the parity checks of one of these codes and $Z$ stabilisers derived from parity checks of the other. One advantage of this type of code is that we can consider decoding of $X$ and $Z$ errors separately, although recent results have shown that in some settings the use of mixed $X$ and $Z$ stabilisers can lead to improved performance [Bonilla Ataides et al., 2021]. Another useful feature of CSS codes is that there is a simple description of their codewords in terms of thier stabilisers. For a CSS code which enocodes a single qubit, the logical $|0\rangle$ state can be written as

$$\left|\overline{0}\right\rangle = \frac{1}{\sqrt{n}} \sum_{i=1}^{n} X_i \left|\mathbf{0}\right\rangle \tag{1.15}$$

where $X_i$ are the $X$ stabiliser generators of the code and $|\mathbf{0}\rangle$ is the all-zeros vector. The logical $|1\rangle$ state can be written

$$\left|\overline{1}\right\rangle = \frac{1}{\sqrt{n}} \sum_{i=1}^{n} \overline{X} X_i \left|\mathbf{0}\right\rangle \tag{1.16}$$

for any logical operator implementation $\overline{X}$ [Calderbank and Shor, 1996, Steane, 1996b].

### 1.2.4 Surface Codes

In this thesis we will focus on a subset of stabiliser codes called topological codes. These are codes with geometrically local stabilisers and logical qubits encoded in non-local degrees of freedom. The most well known and widely studied topological codes are surface codes such as the toric code [Kitaev, 2003] and the closely related planar code. These codes are perhaps most naturally defined and understood through the lens of homology, but an alternative perspective on these codes is as topological phases of matter. In what follows we will explain both of these interpretations and finally describe a more qubit-efficient version of the planar code called the rotated code.

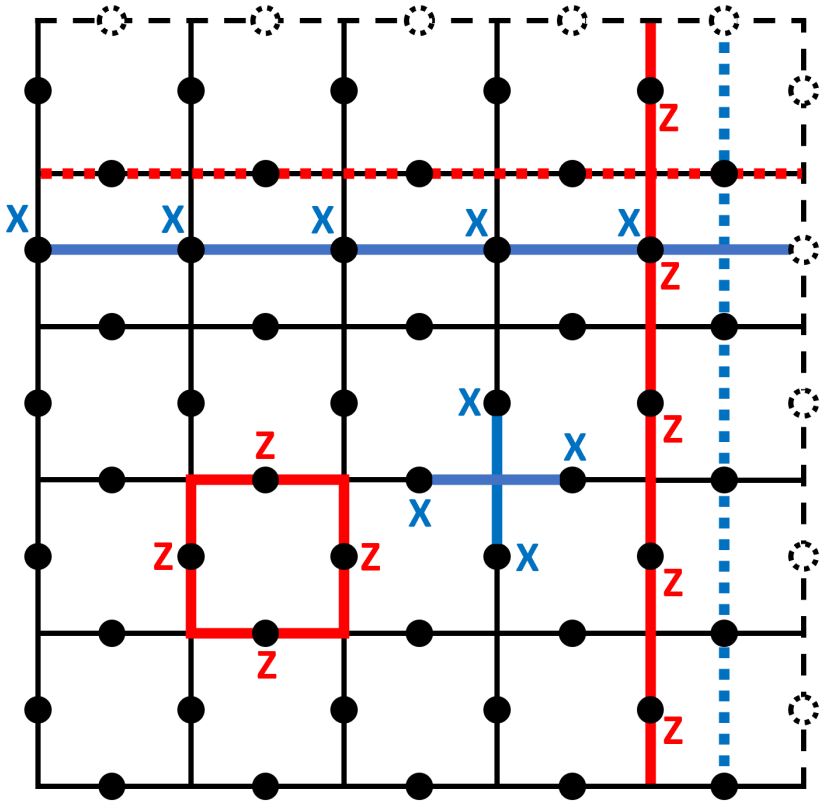



**Figure 1.3:** The distance-5 toric code. The lattice is defined with periodic boundary conditions so the dotted edge on the top (right) is associated with the solid edge on the bottom (left). Qubits are shown as black circles on edges. Weight-4 $X$ and $Z$ stabiliser generators are shown, as well as weight-5 $X$ and $Z$ logical operators for both encoded qubits (logical operators are shown as solid (dotted) lines for the first (second) encoded qubit).

## Homological Perspective

In their original formulation these codes were defined using a square lattice with each edge of the lattice corresponding to a qubit, each vertex corresponding to a weight-4 $X$ stabiliser and each face corresponding to a weight-4 $Z$ stabiliser as shown in fig. 1.3.

The number of logical qubits encoded by a surface code is dependent on the topology of the manifold tessellated by the lattice. Figure 1.3 shows the toric code[1] (surface code defined on a torus) which encodes two qubits. The logical $X$ and $Z$ operators of these qubits are loops of $X$ or $Z$ operators passing around the handle of

[1] Specifically it shows the distance-5 toric code. The distance of the code can straightforwardly be increased by increasing the size of the lattice, so we will refer to a toric code defined on any lattice size as "the toric code" and specify the distance when relevant.

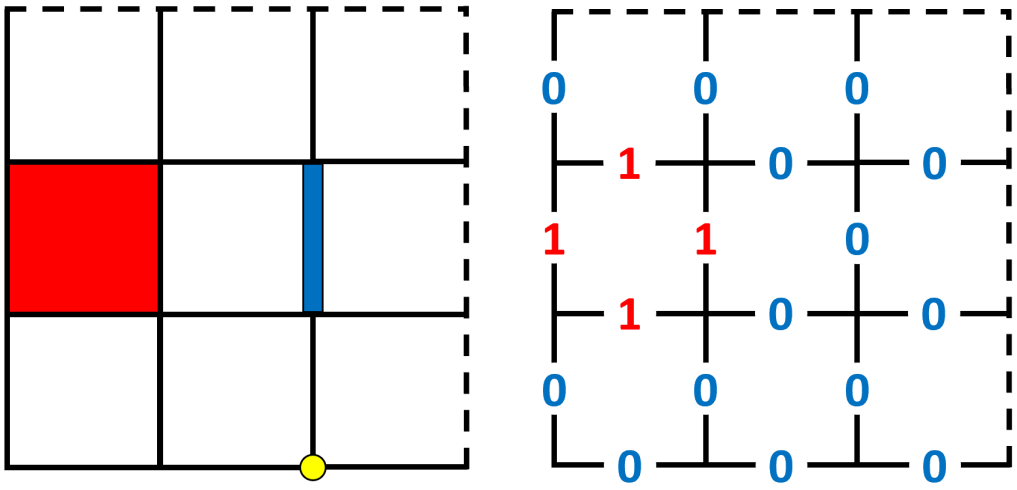



**Figure 1.4:** (a) A 0-cell (yellow), 1-cell (blue) and 2-cell (red). (b) A 1-chain defined by the assignment of 0 or 1 to every 1-cell of a lattice. This can equivalently be thought of as a subset of the edges of the lattice containing only edges assigned 1. This 1-chain corresponds to the boundary of the red 2-cell in (a).

the torus. These loops are called *homologically non-trivial cycles*, but in order to understand this we must first understand some more basic concepts of homology.

In a cellulation of a 2D manifold we have three types of object: 0-cells (vertices), 1-cells (edges), and 2-cells (plaquettes) (fig. 1.4a). The assignment of an element of a group to each $n$-cell is called an $n$-chain, $c_n$. $\mathbb{Z}_2$ homology is most relevant to qubits, so we will only consider the assignment of 0 or 1 (fig. 1.4b). The boundary of an $n$-chain is the $(n-1)$-chain that completely encloses it. For example, the boundary of a plaquette is the edges at its border, while the boundary of an edge is the two vertices at its ends. The boundary map $\delta_n$ is a map from an $n$-chain to its boundary. An $n$-cycle, $z_n$,is an $n$-chain with no boundary, $\delta_n(z_n) = 0_{n-1}$. An $n$-boundary, $b_n$, is an $n$-chain that forms the boundary of an $(n+1)$-chain, $\delta_{n+1}(c_{n+1}) = b_n$. Every $n$-boundary is also an $n$-cycle and so applying the boundary map twice always gives a null chain, $\delta_{n-1}(\delta_n(c_n)) = 0_{n-2}$. More formally, we can define a chain complex

$$C_2 \xrightarrow{\delta_2} C_1 \xrightarrow{\delta_1} C_0 \xrightarrow{\delta_0} \emptyset \tag{1.17}$$

where $C_n$ is the group of $n$-chains.

If two $n$-chains $a_n$ and $c_n$ satisfy $a_n + b_n = c_n$ for some $n$-boundary $b_n$ then we say that $a_n$ and $c_n$ are homologically equivalent. Any two homologically equivalent $n$-chains have the same boundary because

$$\delta_n(a_n + b_n) = \delta_n(a_n) + \delta(b_n) = \delta_n(a_n) = \delta_n(c_n) \tag{1.18}$$

because the boundary map is linear and $\delta(b_n) = 0_{n-1}$ for any $n$-boundary $b_n$. However, not all chains with the same boundary are homologically equivalent. For cycles this difference is captured by the $n^{\text{th}}$ homology group, defined as the kernel of $\delta_n$ modulo the image of $\delta_{n+1}$, or equivalently as the quotient group

$$H_n = \frac{Z_n}{B_n} \tag{1.19}$$

where $Z_n$ is the group of all $n$-cycles and $B_n$ is the group of all $n$-boundaries. For example, the homology groups of the torus are the following:

- The boundary map $\delta_0$ is defined such that $\delta_0(c_0) = 0_{-1}$ for all chains $c_0$, meaning every 0-chain is a 0-cycle. Every 0-chain containing an even number of vertices is the boundary of a 1-chain (a string) so there are two homological equivalence classes: 0-chains with even numbers of vertices and 0-chains with odd numbers of vertices. $H_0$ then has two elements and is isomorphic to $\mathbb{Z}_2$.

- There are four equivalence classes of 1-cycles on a torus: homologically trivial cycles which are the boundaries of plaquettes/set of plaquettes, the two inequivalent cycles passing around the handle of the torus and the product of these two cycles. Elements of the fourth one are products of elements of the second and third so $H_1$ is isomorphic to $\mathbb{Z}_2 \times \mathbb{Z}_2$.

- There are no 3-chains on the torus and so there are no 2-boundaries and $B_2 = 0_2$. The elements of $Z_2$ are boundaryless subregions of the torus, of which there are only two: the trivial membrane (assignment of 0 to all plaquettes) and the entire torus (assignment of 1 to all plaquettes). Therefore $Z_2 = H_2$ as $B_2$ is trivial and so $H_2$ is also isomorphic to $\mathbb{Z}_2$.

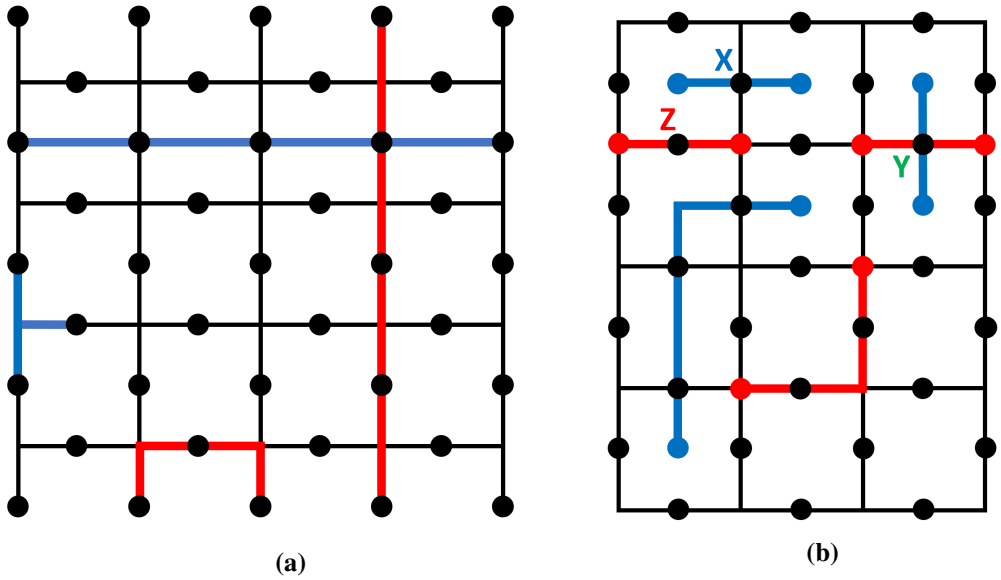



**Figure 1.5:** (a) The distance-5 planar code. This code encodes a single qubit. There are two different types of boundary, "rough" boundaries (top and bottom) which can serve as endpoints for $Z$ strings and "smooth" boundaries (left and right) which can serve as endpoints for $X$ strings. Stabilisers in the bulk are identical to those of the toric code, but there are also weight-3 $Z$ ($X$) stabilisers on the rough (smooth) boundaries as shown in the lower left. The logical operators of the encoded qubit are strings with their endpoints on different boundaries of the same type. (b) Errors in the surface code. Strings of errors only anticommute with stabilisers at their ends (marked with circles). $Y$ errors anticommute with both $X$ and $Z$ stabilisers and can be viewed as crossings of $X$ and $Z$ error strings.

To see the relevance of these objects we can return to fig. 1.3. Each of the three $Z$ operators in this figure is an example of a 1-cycle: the $Z$ stabiliser in the lower left is a homologically trivial cycle equivalent to the boundary of a 2-cell, while the two logical $Z$ operators are both homologically non-trivial cycles. The same thing is true on the dual lattice for $X$ stabilisers and logical operators. In general the number of logical qubits encoded by a surface code on a two-dimensional manifold is equal to the rank of the first homology group, with each generator of this group corresponding to a logical operator on one of the encoded qubits.

Closed manifolds such as the torus are easy to deal with from a theoretical perspective but a planar arrangement of qubits is more practically realistic, so we also wish to consider surface codes on manifolds with boundaries. In a cellulation of an open manifold there are two types of boundary which are interchangably referred

to as rough and smooth, primal and dual or $X$ and $Z$ boundaries. Both boundary types are shown in fig. 1.5a. The qubits on one type of boundary are part of only one $X$ stabiliser, allowing $Z$ strings to terminate at this boundary, while for the other boundary type the reverse is true. Strings with both of their endpoints on the same boundary are homologically trivial whereas strings with their endpoints on different boundaries are not. Therefore a plane with an entirely rough or smooth boundary contains no homologically non-trivial cycles and encodes no qubits, but a plane with alternating boundaries (as in fig. 1.5a) can encode a finite number of qubits (one qubit in the case of fig. 1.5a).

Errors in these codes also have a homological interpretation, as shown in fig. 1.5b. These errors can be viewed as strings on the lattice (or dual lattice for $X$ errors) and the syndrome of an error is given by the boundary of this string. Since stabilisers and logical operators are always cycles they have no boundary, and therefore commute with all stabilisers as expected. Syndromes in these codes do not uniquely identify errors since by definition any two homologically equivalent error strings will have the same boundary and thus the same syndrome. In order to decode these errors we only need to find a string with the correct boundary. We do not require that this string is identical to the original error, only that the product of error and correction operator is a homologically trivial cycle as this corresponds to the application of a stabiliser to the code.

## Condensed Matter Perspective

Surface codes can also be interpreted as topological phases of matter. The degenerate ground state of this phase is the $+1$ eigenstate of all the stabilisers, meaning that errors can be understood as creation operators for quasiparticle excititions in this phase. As we will see in the next section, this perspective on topological codes can be very useful for classifying possible logical operations.

More formally, the Hamiltonian of this phase is

$$H = -\sum_{v \in V} X_v - \sum_{p \in P} Z_p \tag{1.20}$$

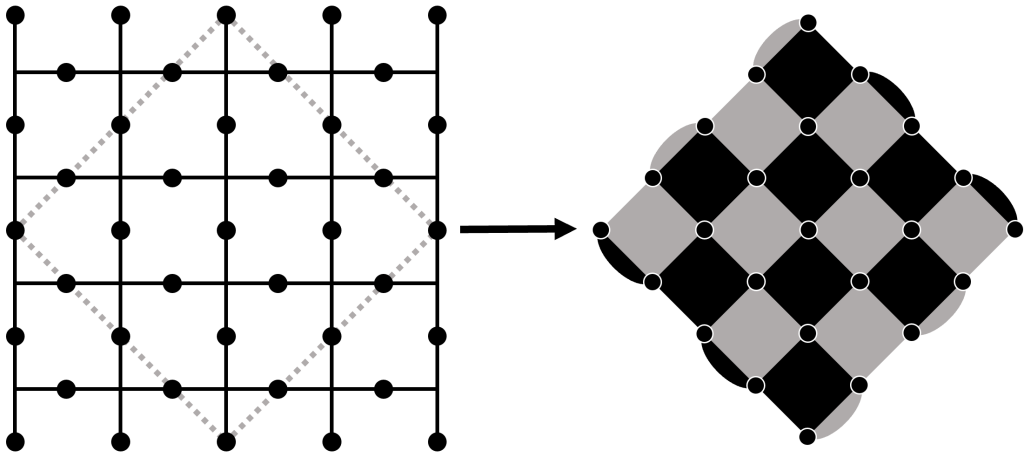


**Figure 1.6:** The distance-5 unrotated (Kitaev picture) and rotated surface codes. $X$ stabilisers in the rotated code are on dark plaquettes while $Z$ stabilisers are on light plaquettes. The two different boundary types in this code are created by the addition of weight-2 $X$ or $Z$ stabilisers to the relevant boundary. The unrotated code uses 41 qubits, while the rotated code only uses 25.

where $V$ ($P$) is the set of all vertices (plaquettes) and $X_v$ ($Z_p$) means $X$ applied to all edges which contain vertex $v$ ($Z$ applied to all edges contained in face $p$). The ground space of this Hamiltonian corresponds to the $+1$ eigenstate of all stabilisers, i.e. the ground state of this phase is the same as the codespace of the code. A string of $X$ ($Z$) operators in the bulk of the code creates a quasiparticle pair on the faces (vertices) at the ends of the string. A string of $Y$ operators creates two $X+Z$ quasiparticle pairs, each of which can itself be thought of as a single quasiparticle excitation. If we also consider the trivial excitation $\mathbf{1}$ then these four types of quasiparticle form an anyon model called the quantum double of $\mathbb{Z}_2$. The specifics of this model, and of anyon models in general, are discussed in more detail in the next chapter.

## The Rotated Surface Code

The original surface codes introduced by Kitaev associate qubits with edges of a lattice (this construction is sometimes referred to as the "Kitaev picture"). There also exists a more qubit-efficient version of the surface code called the rotated surface code [Horsman et al., 2012] where qubits are associated with vertices while $X$ and $Z$ stabilisers are associated with alternating faces of the lattice. A comparison between rotated and Kitaev surface codes of the same distance is shown in fig. 1.6. The

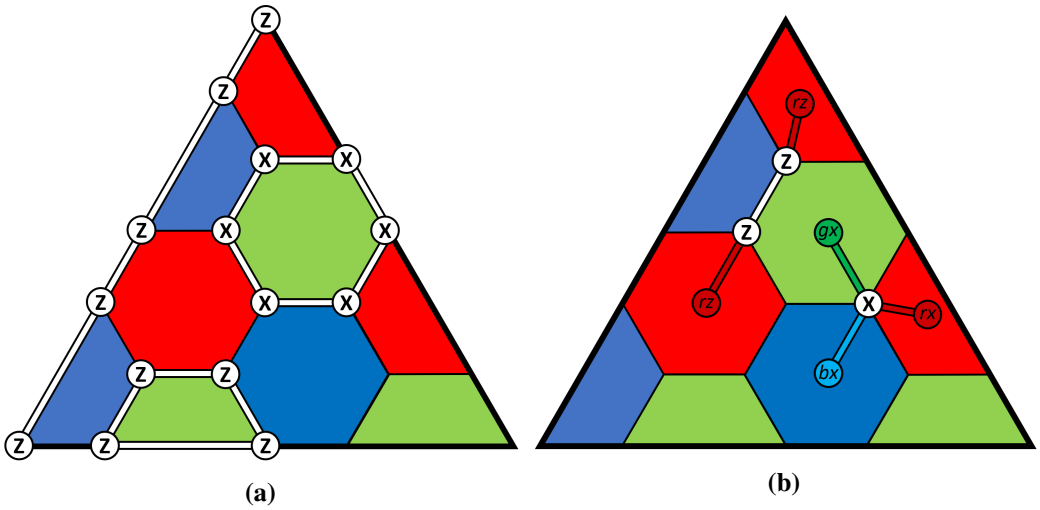


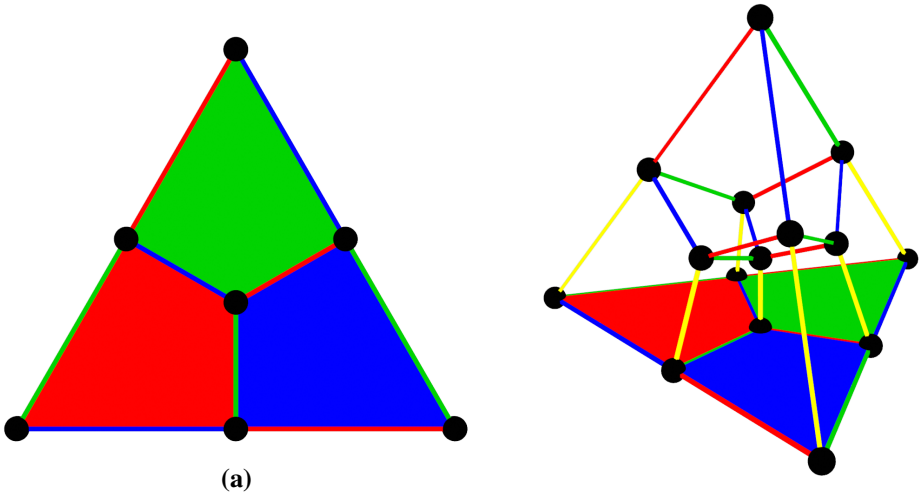

**Figure 1.7:** The distance-5 triangular colour code. (a) Qubits are associated with vertices and each plaquette supports an $X$ and a $Z$ stabiliser. Logical operators for both $X$ and $Z$ are supported on the boundaries of the code (an example of logical $Z$ is shown on the blue-red boundary). (b) Error syndromes in the colour code. Single errors anticommute with one stabiliser generator of each colour, while pairs of errors on the same edge anticommute with two stabiliser generators of the same colour. Syndromes can be described using quasiparticle excitations of the corresponding topological phase.

rotated version of the code uses only 25 qubits, compared to the 41 used by the Kitaev version. Most modern work involving the surface code uses the rotated version, and so we will focus on this version of the code in subsequent chapters.

### 1.2.5 Colour Codes

Colour codes are another family of topological codes which will be relevant in later chapters. Colour codes can be defined for lattices of any dimension $\geq 2$ but in this section we will focus on the 2D and 3D cases [Bombín and Martin-Delgado, 2006, 2007, Kubica, 2018]. This code is defined on a trivalent lattice with faces which are 3-colourable and have even numbers of vertices. A qubit is associated with each vertex of this lattice and an $X$ and a $Z$ stabiliser are associated with each face (fig. 1.7). The fact that faces have even numbers of vertices ensures that $X$ and $Z$ stabilisers from the same face commute, while the trivalency/3-colourability of the lattice ensures that neighbouring faces always meet at an edge, so $X$ and $Z$ stabilisers from different faces also commute.

Error syndromes in the colour code can also be understood as an anyon model



(a) (b)

**Figure 1.8:** (a) A 7-qubit 2D colour code. (b) A 15-qubit 3D colour code. A $Z$ stabiliser is supported on each face and an $X$ stabiliser is supported on each cell. The bottom boundary of this code matches the 2D colour code shown in (a).

of a topological phase. This model contain nine non-trivial bosonic anyons which can be indexed by the colour of plaquette they exist on and the Pauli that created them, e.g. an $X$ error that anticommutes with a stabiliser on a red plaquette creates an $rx$ anyon on that plaquette. There are also six fermionic anyons which can be formed from fusions of the bosonic anyons. This anyon model will be discussed in more detail in the next chapter.

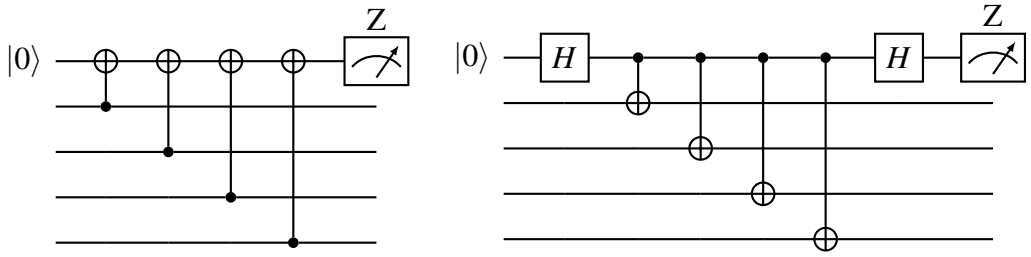
The colour code can be defined on a planar lattice as in fig. 1.7. Each of the three boundaries of this lattice is associated with a particular colour pair and supports both a logical $X$ and $Z$ operator of the code. It can be shown that (up to local unitary operations) this code is equivalent to two copies of the planar surface code [Kubica et al., 2015].

We can also define a three-dimensional version of the colour code. This code can be defined on a lattice with four-colourable cells and qubits on vertices. If an edge connects two $\kappa$ coloured cells then we can colour that edge $\kappa$, meaning that $\kappa$ colour cells will be formed from three colours of edge (the three colours which are not $\kappa$). Each face of the code supports a $Z$ stabiliser and each cell of the code supports an $X$ stabiliser. The 15-qubit code shown in fig. 1.8b has 18 faces and 4 cells, but 4 faces of a given cell can generate the remaining 2 so there are $(18-2\times 4)+4=14$

stabiliser generators and this code encodes a single qubit. An implementation of logical $X$ in this codes is supported on any facet (boundary) of the tetrahedron while an implementation of $Z$ is supported on any edge of the tetrahedron. There also exists a version of the 2D colour code on each one of these facets.

### 1.2.6 Fault Tolerance

In order for reliable quantum computation in the presence of noise we require more than the ability to correct errors on the physical qubits of our code. We must also be able to account for errors in the stabiliser measurements themselves. To understand these errors we must first consider the stabiliser measurement process in more detail. Measurement of stabilisers of arbitrary weight can be acheived with only two-body operations via the use of an ancilla. For example, the following circuits allow measurement of the weight-4 $Z$ and $X$ stabilisers of the surface code.




The term "measurement error" usually refers to an error in the actual measurement operation in these circuits (such that the measurement outcome is 1 when it should be $-1$ or vice versa). These errors cause problems during decoding since the decoder will be passed an incorrect syndrome and thus could return an incorrect "correction" operator which may actually introduce more errors into the code. The standard approach to dealing with these errors in codes such as the surface and colour code is to repeat the stabiliser measurements many times before applying a correction. Usually the measurements are repeated $d$ times (where $d$ is the code distance) and we say that the resulting error correction process has distance $d$ against both data qubit errors and measurement errors. More complex codes such as the three-dimensional surface and colour codes are capable of *single-shot error correction* [Bombín, 2015, Quintavalle et al., 2021] which allows for the correction of measurement errors after

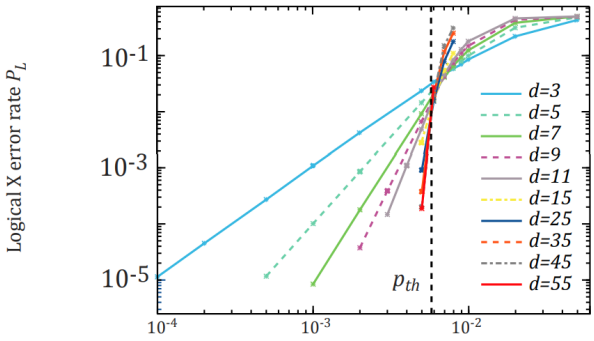



**Figure 1.9:** Numerical demonstration of a threshold for the 2D surface code with noisy data qubits and noisy measurement circuits (from Fowler et al [Fowler et al., 2012]). The simulations show a threshold of $\sim 0.57\%$ for a minimum-weight perfect matching decoder in this setting. The x-axis of the plot shows the "per-step error rate", which gives the probability of an error occuring following each operation at each timestep of the measurement circuit.

only a single round of stabiliser measurements. This process will be discussed in more detail in chapter 3.

There are also other kinds of errors which can occur in the measurement circuits shown above. For example, an $X$ error on the ancilla in the second circuit can spread through the CNOT gates and cause multiple $X$ errors on the data qubits of the code. If this error occurs between the second and third CNOT gates then it will spread to a two-qubit $X$ error on the third and fourth qubits. These errors are sometimes called "hook errors" Dennis et al. [2002]. Errors occuring earlier in the circuit can cause three or four-qubit errors but these are equivalent to one or zero-qubit error up to composition with the $X$ stabiliser that the circuit is measuring, so the two qubit hook error is the worst case in the surface code. In the colour code we have weight-6 stabilisers and so weight-3 hook errors can occur. In general codes with low stabiliser weights are desirable as this minimises the possible impact of hook errors.

It has been shown that, provided error rates are low enough, the surface code can operate reliably even in the presence of measurement errors. Figure 1.9 (taken from work by Fowler et al [Fowler et al., 2012]) shows the results of numerical simulations of the performance of the surface code when subject to all of the previously described error types, as well as more general gate errors in the measurement circuits and

errors in the preparation of the state of the ancilla qubit. These simulations use a minimum-weight perfect matching (MWPM) decoder to pair up the endpoints of error strings and find correction operators, and show the existence of a threshold error probability of around $\sim 0.57\%$. Below this threshold the logical error rate can be arbitrarily suppressed by increasing the size of the code.

Thresholds for more complicated codes and decoding processes will be discussed in more detail in chapter 3.

## 1.3 Fault-Tolerant Logic and Code Deformation

In the previous section we discussed the ability of quantum error correcting codes to protect qubits from noise. However, in order to perform computation with the encoded qubits we also require the ability to manipulate the stored information. This is more challenging than performing computational operations on unencoded qubits since these operations must be performed in a way which does not compromise the fault-tolerance of the code. The most straightforward method of achieving this is via the use of *transversal* gates, which do not entangle physical qubits belonging to the same code. For example, the application of a single-qubit gate to each of the $n$ qubits of a code constitutes a transversal operation, as does the application of $n$ two-qubit gates between pairs of qubits from different codes. However, applying $n/2$ two-qubit gates between pairs of qubits in the same code is not a transversal operation. Transversal operations are inherently fault-tolerant since they cannot increase the weight of errors within a code, but the computational power of these operations is limited.

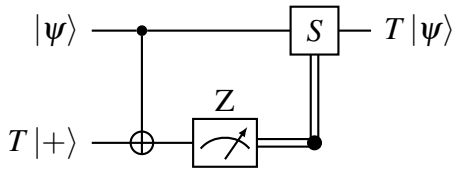
The Eastin-Knill theorem tells us that no error correcting code which can detect arbitrary errors on individual qubits has a universal set of transversal gates[2][Eastin and Knill, 2009]. Beyond this there are a number of no-go theorems restricting the power of transversal gates in more specific code families, as well as restrictions on the power of various generalisations of transversal gates such as locality-preserving logical operators and general unitary operators [Eastin and Knill, 2009, Zeng et al.,

[2] A set of gates is considered to be universal for quantum computation if any unitary operation can be approximated arbitrarily well using only gates from the set

2011, Bravyi and König, 2013, Jochym-O'Connor et al., 2018, Burton and Browne, 2020, Webster et al., 2021, Webster and Bartlett, 2020].

Logical gate implementations which are not subject to to these restrictions have been the subject of a significant amount of research. Perhaps the most popular approach uses so-called "magic states" and was first proposed by Bravyi and Kitaev [Bravyi and Kitaev, 2005]. This approach to fault tolerant logic consists of two steps: state distillation and state injection.

- **State Injection:** Consider the following circuit




  where $T$ is the rotation $\mathrm{diag}(1, e^{i\pi/4})$. This circuit takes in an arbitrary state $|\psi\rangle$ and a state $T|+\rangle$ (a $T$-state) and uses the latter to apply the gate $T$ to the former. This means that if we have access to a supply of high-fidelity $T$-states and can implement the operations in this circuit fault-tolerantly then we can also implement $T$ gates fault-tolerantly. This is significant for two reasons. Firstly, the only operations in this circuit are Pauli measurements and Clifford gates (gates which map Paulis to Paulis). Both of these classes of operations can be performed fault-tolerantly in 2D topological stabiliser codes such as the surface and colour codes. Secondly, the $T$ gate is a non-Clifford gate and any generating set of the Clifford group plus a single non-Clifford gate is known to constitute a universal gate set [Nebe et al., 2006]. Therefore, given a method of preparing high-fidelity $T$-states that requires only fault-tolerance of Clifford operations we can fault-tolerantly implement a universal gate set. This is exactly what state distillation gives us.

- **State Distillation:** This is a process which takes as input a large number of low-fidelity copies of a state and outputs a smaller number of higher fidelity copies. Importantly, this distillation process requires only Clifford operations. There exist a wide variety of state distillation schemes with various advantages

and disadvantages in terms of thresholds, efficiency and so on, but these schemes can be fairly complex and reviewing them is not the aim of this thesis so we refer interested readers to the following sources [Bravyi and Kitaev, 2005, Litinski, 2019, Campbell and Howard, 2017, Haah et al., 2017].

Magic state distillation requires a large amount of additional hardware in order to produce the many copies of states needed for the distillation procedure and so other methods of achieving universal logic have also been investigated. These include the code deformation techniques described previously, although not all of these techniques provide universality and some instead provide more convenient implementations of gates that also have transversal implementations. In the remainder of this section we discuss some simple examples of code deformation techniques and explain their usefulness.

### 1.3.1 Defect Encodings and Braiding

In section 1.2.4 we saw that logical qubits in the surface code could be defined using lattice patches with mixed boundary types. Another method of defining logical qubits is through the use of lattice defects. One type of defect is a puncture, which corresponds to a region of removed stabilisers in the interior of the code (i.e. we choose to no longer measure these stabilisers). Figure 1.10 a) shows two puncture encoded qubits in a surface code, one corresponding to a removed $X$ stabiliser and one to a removed $Z$ stabiliser. As discussed previously, the number of encoded qubits in a stabiliser code is equal to the number of physical qubits minus the rank of the stabiliser group. Each of the punctures in fig. 1.10 a) is created by the removal of a single stabiliser generator while the number of physical qubits in the code is unchanged, so each increases the number of encoded physical qubits by 1. The logical operators for these new qubits are also shown, and can be seen to correspond to cycles which are homologically equivalent to the removed stabiliser or to strings which would previously have been detected by the removed stabiliser, thus creating two new logical Pauli operators for each removed qubits as we would expect. Notice that the distance of these new logical qubits depends on two things, the size of the puncture and its distance from the code boundary (or from another puncture of the

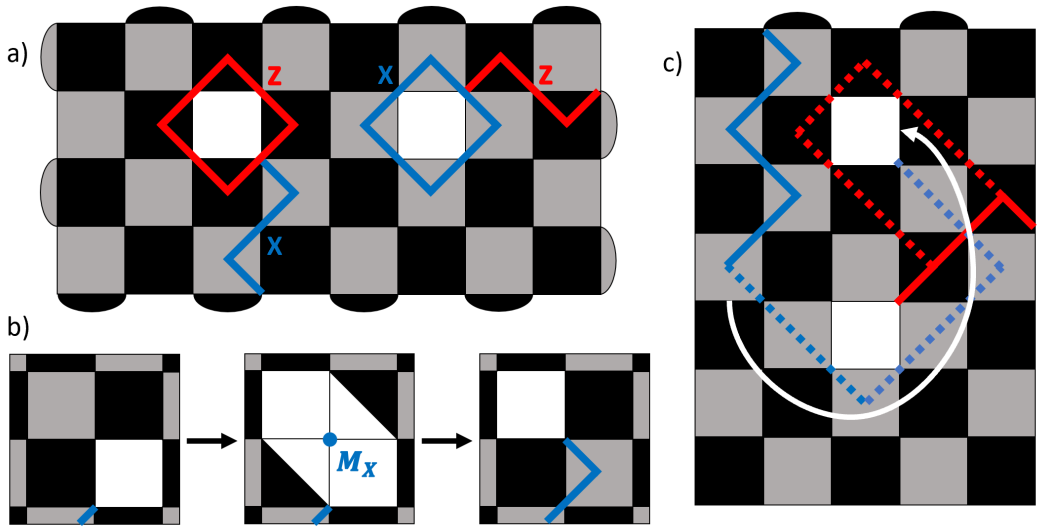



**Figure 1.10:** Puncture encodings in the 2D surface code. a) shows $X$-type and $Z$-type punctures corresponding the removal of $X$ and $Z$ stabiliser generators respectively. Each puncture encodes a single qubit and pairs of associated logical operators are shown for each. b) Growing, shrinking and moving punctures in a subregion of the code. Physical qubits completely contained within the puncture are measured and removed from the code. When a puncture moves, its associated logical operators move with it. c) Braiding a $Z$-type puncture around an $X$-type puncture to perform a logical CNOT gate. Pre-braid logical operators are shown by solid lines and changes caused by the braid are shown by dashed lines.

same type, since strings connecting the two also correspond to logical operators).

Both of these properties can be modified as shown in fig. 1.10 b), where we show how punctures can be grown and shrunk, and how this process can be used to move them around the lattice. Notice that if we grow the puncture to be large enough that there are qubits within it which are not part of any stabilisers then we must measure out these qubits to remove them from the code. In the figure we grow an existing puncture by removing one stabiliser generator and one physical qubit, meaning the number of encoded logical qubits remains the same. When we shrink the puncture we add a generator and a physical qubit. Notice also that as we move this puncture through the code the connected logical operator moves with it. This can be used to perform braiding operations with punctures, as in fig. 1.10 c). This figure shows a case where we have braided a $Z$-type puncture (i.e. corresponding to the removal of a $Z$ stabiliser generator) around an $X$-type puncture. This causes the logical $X$

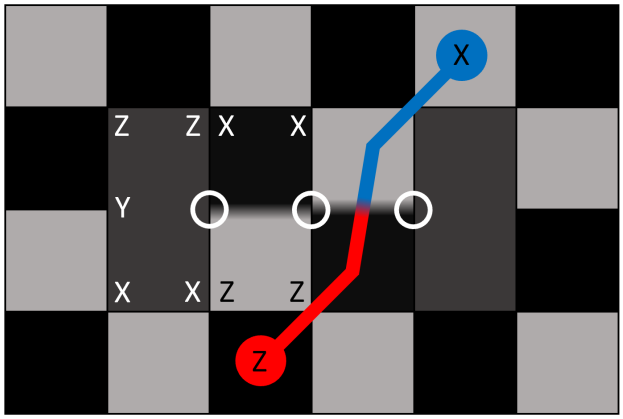



**Figure 1.11:** A domain wall and pair of twists in the 2D surface code. Qubits are on vertices. Dark plaquettes correspond to $X$ stabilisers and light plaquettes to $Z$ stabilisers. Each twist corresponds to a weight five stabiliser which includes one $Y$. $X$ and $Z$ stabilisers and excitations are exchanged by the domain wall. The domain wall can be created in a regular surface code by measuring out the three circled qubits and measuring the new stabilisers.

operator of the $Z$-type puncture to become wrapped around the $X$-type puncture, and this string is homologically equivalent to a product of the logical $X$ operators of the two punctures. Similarly, the logical $Z$ of the $X$-type puncture gets "caught" on the $Z$-type puncture during the braid and the resulting string is homologically equivalent to a product of logical $Z$ operators for the two punctures. In this way we can see that the braiding operation maps $X \otimes I \rightarrow X \otimes X$ and $I \otimes Z \rightarrow Z \otimes Z$. It does not affect $I \otimes X$ or $Z \otimes I$, and this is consistent with the logical action of the CNOT gate on a pair of qubits.

Puncture encodings and braidings can also be understood in terms of the toric code anyon model. We can think of the $Z$-type puncture as a box that can hold $X$-type quasiparticles. The logical $X$ for this puncture transfers a quasiparticle into/out of this box and the logical $Z$ operator detects the presence or absence of these quasiparticles (corresponding to logical $|1\rangle$ or $|0\rangle$). The same is true for the $X$-type punctures, although the roles played by $\overline{X}$ and $\overline{Z}$ are reversed in this case, so the presence of a $Z$-type quasiparticle corresponds to $|-\rangle$ while the absence corresponds to $|+\rangle$. Strings of $X$ and $Z$ operators anticommute, meaning that if we exchange an $X$-type quasiparticle with a $Z$-type quasiparticle we acquire a phase of $-1$. CNOT acts

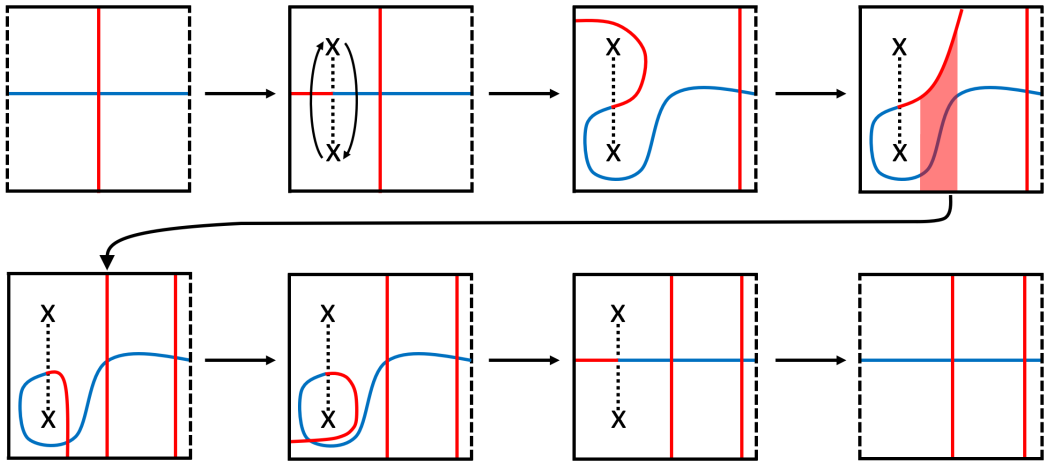


**Figure 1.12:** A twist braiding operation that can be used to implement a logical $S$ gate in a planar surface code [Brown et al., 2017]. Code deformation operations are used to bring a pair of twists and a domain wall from the boundary into the bulk and then these twists are exchanged and merged back into the boundary. In the middle sequence of images we see how to untangle the logical operators from around the twist by composing them with stabilisers such as the shaded red region in the fourth image. Step $6 \rightarrow 7$ uses the fact that a string that loops twice around a twist is also a stabiliser of the code.

as identity on the states $|0+\rangle$, $|0-\rangle$ and $|1+\rangle$ (i.e. the states where we have 0 or 1 quasiparticles) while $\text{CNOT}|1-\rangle = -|1-\rangle$ and so CNOT can be implemented by this exchange.

Another type of defect which can encode logical qubits is referred to as a "twist". These are endpoints of domain walls or lattice dislocations in the code, and braids using twists can give access to gates not accessible via braiding punctures. An example of a domain wall and two twists in the 2D surface code is shown in fig. 1.11. $X$ and $Z$ are exchanged by the domain wall, meaning stabilisers which straddle it consist of two $X$ and two $Z$ operators and error strings which cross it can create one $Z$ and one $X$ quasiparticle instead of a pair of identical quasiparticles [Bombín, 2010].

$X$ ($Z$) boundaries in the 2D surface code can be thought of as combinations of $Z$ ($X$) boundaries and one of these $X$-$Z$ exchanging domain walls, meaning that the corners of this code (where $X$ and $Z$ boundaries meet) correspond to twists [Brown et al., 2017]. Code deformation operations can be used to move these twists into the bulk of the code where they can be used in braiding operations to modify the encoded

logical information. An example of such a braiding operation is shown in fig. 1.12. Here we can see how the exchange of two twists maps the logical $X$ operator of the code to a product of $\overline{X}$ and $\overline{Z}$ but has no effect on the logical $Z$ operator. This is consistent with the action of a logical $S$ gate on the encoded qubit. Note that the 2D surface code does not admit a transversal implementation of $\overline{S}$, and nor can this gate be implemented via braiding of punctures.

### 1.3.2 Lattice Surgery

Lattice surgery [Horsman et al., 2012] is another method for implementing a logical CNOT gate, although it uses surface code patches with mixed boundaries (as discussed above) rather than puncture-encoded qubits. Such patches admit a transversal CNOT gate but this gate cannot be applied without either non-local couplings or a 3D architecture as a CNOT gate must be performed between each qubit in the first code and the corresponding qubit in the second code. In contrast, the lattice surgery CNOT requires only edge-to-edge contact of code patches. It consists of two basic operations, referred to as a merge and a split, which can be performed between pairs of $X$ or $Z$ boundaries. Figure 1.13a shows examples of these operations for a $Z$ boundary. In the first image we have two distance-3 surface code patches with adjacent $Z$ boundaries. In the second image we have measured two new $X$ stabilisers, resulting in these two codes being merged into a single code encoding only a single logical qubit. Two of the $Z$ boundary stabilisers have been merged into a single plaquette stabiliser and so we have a net increase of one stabiliser generator, consistent with the loss of one encoded qubit. The product of the two new $X$ stabilisers is a weight-6 operator equivalent to the product of two logical $X$ operators from the original (separate) codes, and so measurement of these stabiliers can be interpreted as a logical $XX$ measurement. The third image shows a split operation, where we can divide the code in the second image back into two separate codes via measurement of the original $Z$ boundary stabilisers. Note that the two encoded qubits will still be entangled after this split.

Figure 1.13b shows an arrangement of three code patches which can be used for a lattice surgery CNOT. The states of the three qubits are labelled $C$, $A$, and

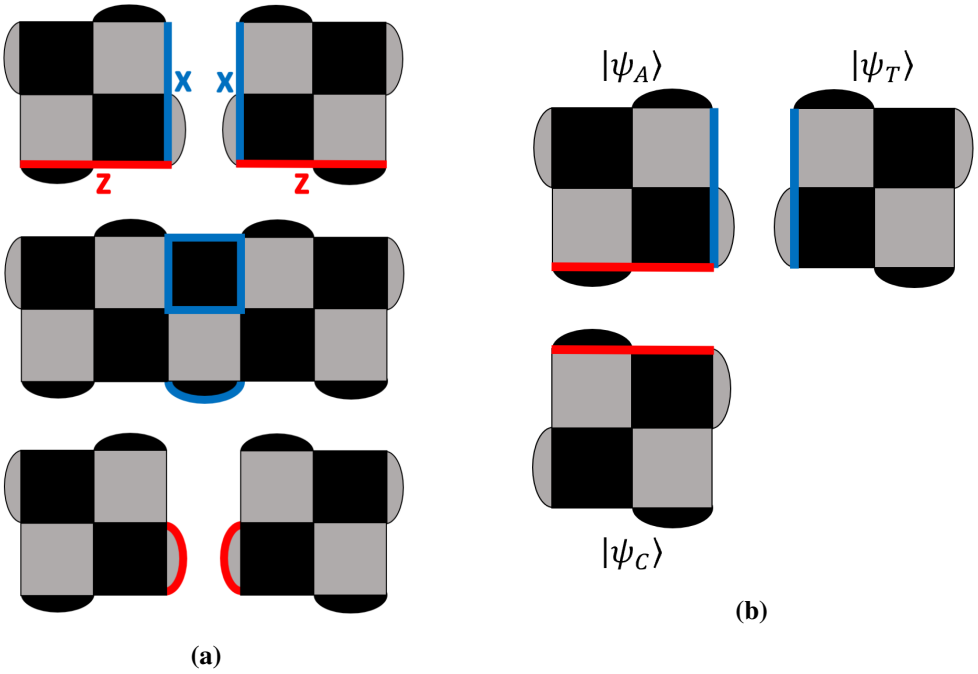



**Figure 1.13:** Lattice surgery in the 2D surface code. (a) Merge and split operations transforming a pair of surface codes into a single code and then back again. These operations are performed entirely via measurement, with new stabilisers measured at each step shown with thick coloured outlines. (b) An arrangement of three surface codes allowing for the implementation of a logical CNOT via lattice surgery. The gate is implemented by merging and then splitting the control and ancilla codes, then merging the ancilla and target codes.

$T$ for control, ancilla and target. It can be shown that performing an $X$ boundary merge and split between the control and ancilla, and then a $Z$ boundary merge between the ancilla and target implements a logical CNOT between the control and target qubits. The work in this thesis does not deal directly with lattice surgery and so we do not show the full calculation here and instead refer interested readers to [Horsman et al., 2012]. Lattice surgery between 2D and 3D surface codes has also been proposed [Vasmer and Browne, 2019].

### 1.3.3 Dimension Jumping

The final technique we will discuss is termed "dimension jumping" and provides a way of fault-tolerantly switching between 2D and 3D versions of a topological code. It was first proposed for the 2D and 3D colour codes [Bombín, 2016] and that is the setting in which we will discuss it here, although the surface code case

will be discussed in chapter 3. Additionally, the original formulation of dimension jumping uses a more complex version of the 3D colour code called the gauge colour code, but for simplicity we describe an alternate version of this process using the regular (stabiliser) colour code. To understand this procedure we must first observe that each boundary of the 3D colour code supports a 2D colour code, as shown in fig. 1.8. Notice also that all $X$ stabilisers of the 3D code commute with all $Z$ stabilisers of the 2D code and there are implementations of the logical operators of the 3D code supported entirely in the 2D code. The dimension jump is a transfer of logical information from this boundary into the bulk, or vice versa.

The 2D $\rightarrow$ 3D expansion is fairly simple. We begin with a 2D code in state $|\overline{\psi}\rangle$ and all other qubits of the 3D code prepared in the state $|+\rangle$. We then measure all of the $Z$ stabilisers of the 3D code (some of which are stabilisers of the 2D code). Assuming we had no errors in the 2D code we will get $+1$ outcomes from all of the $Z$ stabilisers which are part of the 2D code and random outcomes from the others. We can then find a correction operator (which will be supported only on qubits which are not part of the 2D code) which maps us to the state $|\overline{\psi}\rangle$ in the codespace of the 3D code.

To collapse back from 3D to 2D we first make single-qubit $X$ measurements of all qubits which are not part of the 2D code and then apply a correction to the 2D code based on the outcome of these measurements. The calculation of this correction is discussed in depth for the 3D surface code in chapter 3 and a similar process can be applied to the colour code, but in short it involves finding a $Z$ stabiliser of the 3D code which is supported on all the qubits where our $X$ measurements returned $-1$ and then applying the restriction of this stabiliser to the 2D code. This differs from the process discussed in the original proposal of this technique [Bombín, 2016] which uses the 3D gauge colour code instead of the stabiliser code discussed here.

Unlike braiding and lattice surgery, dimension jumping by itself does not perform a logical operation. Its utility comes from the fact that the 2D colour code admits transversal implementations of the full Clifford group, while the 3D colour code admits a transversal implementation of the non-Clifford gate $T$. We can

therefore acquire a universal gate set by switching between the two codes as needed.

## 1.4 Summary

We have discussed various approaches to fault-tolerant logic in stabiliser codes including a number of code deformation techniques. In what follows we will investigate applications of these techniques in more detail. In chapter 2 we discuss in detail an alternate type of defect encoding scheme that uses twist defects rather than punctures. We review existing work using these defects and then generalise some of these results, obtaining braiding relations for a large class of these defects. In chapter 3 we discuss a recently proposed decoding technique called just-in-time decoding which is designed to be used in conjunction with version of the dimension jumping process discussed above. We explain this procedure in detail and then obtain numerical evidence for a threshold. Chapter 4 is not directly concerned with code deformation techniques, and instead focuses on analysis of a particularly complex type of error that can occur during the procedure discussed in chapter 3. A related type of error has previously been studied in the 3D colour code and we generalise these results to the 3D surface code and show that they have a more natural interpretation in this setting.

## Chapter 2

# Fusion and Braiding of Twists in Stacked Surface Codes

As discussed in the previous chapter, logical information in topological codes can be encoded in lattice defects and logical operations can be performed by braiding these defects. Examples of defects include punctures and twists, with the latter being of particular interest as they can be used to reproduce the braiding relations of anyons not in the anyon model of the code and so can allow access to additional logical gates. For example, the S gate cannot be implemented transversally in the 2D surface code (outside of folded surface code constructions such as [Moussa, 2016]) but can be implemented via braiding of twist defects [Brown et al., 2017].

A recent paper by M. Kesselring et al [Kesselring et al., 2018] fully categorises the twists of the 2D colour code, sorting them into nine conjugacy classes. In light of this result it seems natural to ask what gates we can implement via the braiding of these twists. In this chapter we answer this question for at least some of the colour code twists. In [Brown et al., 2017] the fact that braiding twists produces an S gate is shown by considering the action of this braid on the logical operators of the code (this same proof is shown in fig. 1.12). However, the same result can be obtained by considering the twists as (Ising) anyons and analysing their braiding relations, as in [Bombín, 2010]. Formally the twists in a topological code are described by G-crossed braided tensor categories [Barkeshli et al., 2014] and cannot in general be considered as anyons. We will discuss below the cases in which neglecting the

full G-crossed category treatment of these defects is permissable and we will see that three of the nine conjugacy classes of colour code twist are examples of cases where twists can be analysed using anyonic models. These models are members of a hierarchy of anyonic models generalising the standard Ising anyon model used to study twists in the surface code.

This chapter is organised as follows: in section 2.1 we present a short overview of anyon fusion and braiding relations and twists in topological codes, touching briefly on the G-crossed braided tensor category formalism and the occasions when it is acceptable to disregard it. This first section is not original work, but serves as a chapter-specific literature review. We then define a hierarchy of "extended Ising models" in section 2.2 and discuss the general fusion and braiding relations for these models in section 2.3 and section 2.4 respectively. In section 2.5 we clarify the correspondence between these relations and the possible logical operations that can be performed using these anyons. Finally, in section section 2.6 we discuss how general models in this hierarchy can be realised in stacks of 2D surface codes with special attention given to the case of the 2D colour code. The original work described in this chapter was presented previously in [Scruby and Browne, 2020].

## 2.1 Anyons and Twists

In this section we briefly review the theoretical background necessary for the rest of the chapter. In section 2.1.1 and section 2.1.2 we provide an overview of anyon fusion and braiding relations and in section 2.1.3 we present a similar discussion regarding twists in topological codes and briefly touch on the category theory formalism that describes these objects.

### 2.1.1 Fusion and Braiding

There exist a wide variety of ways to describe anyon models. They can be described in terms of topological charges [Bombín, 2010], unitary braided tensor categories [Barkeshli et al., 2014] and through the lens of conformal field theory [Bais and Slingerland, 2009, Francesco et al., 1997]. For our purposes the topological charge description is largely sufficient, although we will very briefly use the category-

theoretic approach when discussing twists in section 2.1.3. This means that we have some finite number of anyonic species, each possessing a unique label and topological charge. The anyons obey a set of fusion and braiding relations. Fusion relations are generally written in the form [Pachos, 2012]

$$a \times b = \sum_c N_{ab}^c c \tag{2.1}$$

where $N_{ab}^c$ is an integer counting the number of ways anyons $a$ and $b$ can fuse into $c$. For a given charge $a$, if, for every choice of $b$, $N_{ab}^c$ is non-zero for at most a single charge $c$ (i.e. the fusion of $a$ with any other anyon has only one possible result) then we say that $a$ is Abelian. Otherwise it is non-Abelian. All anyon models must contain a unique vacuum charge $\mathbf{1}$ such that $a \times \mathbf{1} = a$. Additionally, each charge $a$ in the model must have a unique inverse $\bar{a}$ with which it can fuse to the vacuum in a unique way ($N_{a\bar{a}}^{\mathbf{1}} = 1$).

The total anyonic charge within a given region is a topological invariant and so cannot be altered by operations within this region. However, through alterations to anyon fusion order and intermediate fusion outcomes we can arrive at this same total charge via different paths, called fusion channels. This gives rise to the notion of an anyonic fusion space with dimension equal to the number of possible fusion channels. The quantum dimension of an anyon is defined to be

$$d_a^2 = \sum_b N_{a\bar{a}}^b d_b \tag{2.2}$$

meaning that $d_a = 1$ for all Abelian anyons and $d_a > 1$ for non-Abelian anyons. The dimension of the fusion space of $N$ anyons $a$ grows asymptotically as $(d_a)^N$ in the limit of large $N$. Clearly if we wish to use anyons in quantum computation then only those models which contain non-Abelian anyons are of interest to us.

Changes of basis in the fusion space can be described using F-moves,

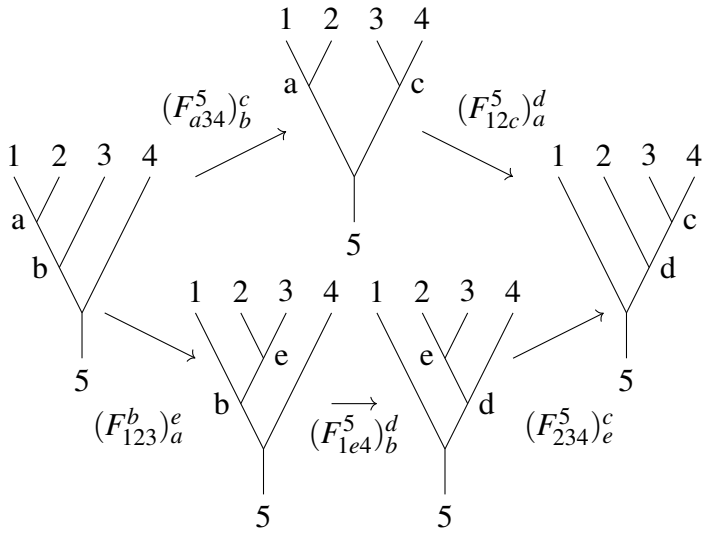
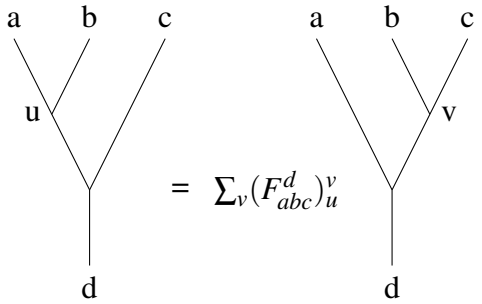



**Figure 2.1:** Diagrammatic representation of the pentagon equation [Pachos, 2012]. Different sequences of F-moves that have the same start and end point must be equivalent.




, (2.3)

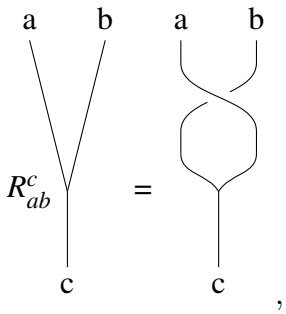
where $u$ ($v$) can be any of the fusion outcomes of $a$ and $b$ ($b$ and $c$). More generally we should include additional indices describing the precise fusion channel by which $a$ and $b$ ($b$ and $c$) fuse to $u$ ($v$) but in what follows we will only consider fusion rules with $N^c_{ab}$ equal to either 1 or 0 and so such generality is unnecessary. The F-matrices associated with an anyon model can be found from the fusion rules by solving the pentagon equation [Pachos, 2012]

$$(F^5_{12c})^d_a (F^5_{a34})^c_b = \sum_e (F^d_{234})^c_e (F^5_{1e4})^d_b (F^b_{123})^e_a \tag{2.4}$$

which can be understood diagramatically in fig. 2.1.

Logical operations on the fusion space can be performed via the braiding of

anyon pairs. This is an operation




, (2.5)

which exchanges the positions of anyons $a$ and $b$ which fuse to $c$. $R^c_{ab}$ will be a phase if $a$ and $b$ have only a single possible fusion outcome, and a diagonal matrix indexed by $c$ if there are multiple possible outcomes. The charges $a$ and $b$ and the outcome of their fusion $c$ are left unchanged by the braiding in accordance with the fact that anyonic charges cannot be modified through local operations. However, the fusion outcome of $a$ or $b$ with a third anyon may be modified by this braid. If the F-matrices for a model are known then we can find the R-matrices for that model using the hexagon equation

$$R^c_{13}(F^4_{213})^c_a R^a_{12} = \sum_b (F^4_{231})^c_b R^4_{1b} (F^4_{123})^b_a. \tag{2.6}$$

As with the pentagon equation, the hexagon equation can more easily be understood when presented diagramatically as in fig. 2.2.

### 2.1.2 Examples

Two anyon models of central importance in our work are the quantum double of $\mathbb{Z}_2$ [Bombín, 2010] and the Ising anyons [Pachos, 2012]. The former is an Abelian model with charges $\mathbf{1}, e, m$ and $\varepsilon$, fusion rules

$$\begin{gathered} e \times e = m \times m = \varepsilon \times \varepsilon = \mathbf{1}, \\ e \times m = \varepsilon, \;\; e \times \varepsilon = m, \;\; m \times \varepsilon = e \end{gathered} \tag{2.7}$$

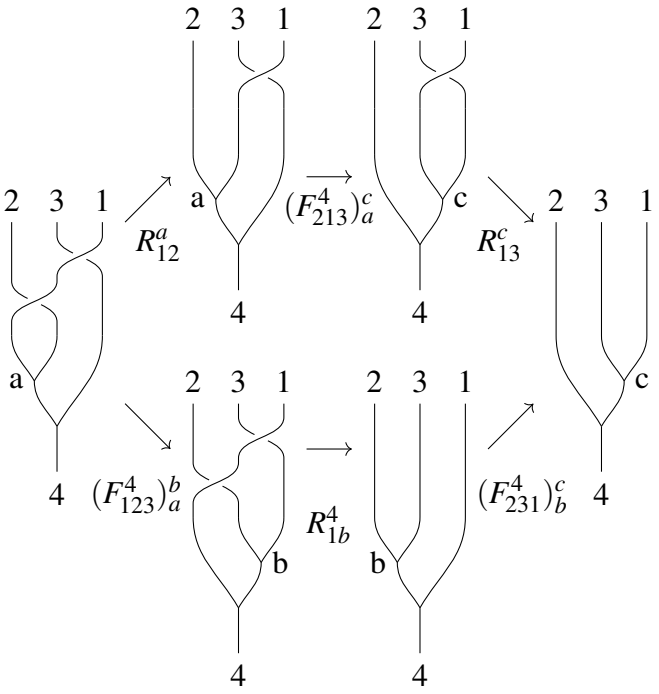



**Figure 2.2:** Diagrammatic representation of the hexagon equation. As with the pentagon equation, different sequences of F and R-moves with the same start and end points must be equivalent

and braiding relations

$$\begin{aligned} R_{ee} = R_{mm} = 1, \;\; R_{\varepsilon\varepsilon} = -1, \\ R_{em}R_{me} = R_{e\varepsilon}R_{\varepsilon e} = R_{m\varepsilon}R_{\varepsilon m} = -1. \end{aligned} \tag{2.8}$$

This model describes the excitations that arise in the toric code, with $e$ and $m$ anyons corresponding to $X$ and $Z$ errors and $\varepsilon$ corresponding to a combination of the two (i.e. a $Y$ error). It also possesses a symmetry: we can exchange the $e$ and $m$ charge labels without affecting any of the fusion or braiding relations.

In contrast, the Ising anyon model is non-Abelian. It contains three charges: $\mathbf{1}, \psi$ and $\sigma$. The Ising anyon fusion rules are

$$\psi \times \psi = \mathbf{1}, \;\; \psi \times \sigma = \sigma, \;\; \sigma \times \sigma = \mathbf{1} + \psi \tag{2.9}$$

and the F matrix for the fusion of three $\sigma$s is

$$F^{\sigma}_{\sigma\sigma\sigma} = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \tag{2.10}$$

and all other $F^{d}_{abc}$ are arbitrary phases. The braiding relations are

$$R_{\sigma\sigma} = e^{-i\pi/8} \begin{pmatrix} 1 & 0 \\ 0 & i \end{pmatrix} \tag{2.11}$$
$$R_{\psi\psi} = -1, \;\; R_{\psi\sigma}R_{\sigma\psi} = -1.$$

### 2.1.3 Twists in Topological Codes

We noted above that the quantum double of $\mathbb{Z}_2$ is symmetric under exchange of $e$ and $m$ charges. We can consider a domain wall which applies precisely this symmetry, achievable in the toric code via a line of modified stabilisers each containing two $Z$ and two $X$ operators as shown in fig. 1.11 [Brown et al., 2017, Bombín, 2010]. An $X$ error moved across such a domain wall will be transformed to a $Z$ error and vice versa. The end points of domain walls such as this are called twists and are formally described by G-crossed braided tensor categories [Barkeshli et al., 2014]. We now give a very brief outline of some of the basic ideas of this formalism. This will be limited to the minimum details required for drawing a connection between twists and anyons and readers interested in a rigourous mathematical description of this formalism should refer to the sources cited.

An anyon model can be described by a unitary braided tensor category $\mathcal{C}_0$ which has charges $a_0$ and a (possibly trivial) symmetry group $G$. The elements of $G$ are labelled $g$ and correspond to the symmetries of the anyon model. The identity element of this group is labelled 0. The action of $g$ on $\mathcal{C}_0$ is an invertible map from $\mathcal{C}_0$ to itself. In a physical realisation of this anyon model each $g$ will correspond to a twist and braiding an anyon around this twist will apply the symmetry $g$ to that anyon, with this action denoted as ${}^{g}a_0$. The topological charge of a twist can be measured by braiding it with an anyon $a_0$ which is invariant under the symmetry $g$

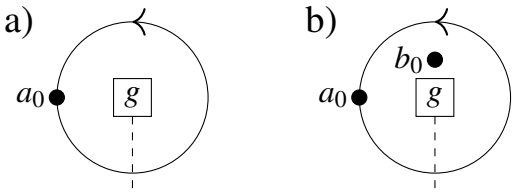



**Figure 2.3:** Braiding an anyon in a loop around a twist to measure the enclosed topological charge. $a_0$ is invariant under the symmetry $g$. cases a) and b) are distinguishable only if $a_0$ braids non-trivially with $b_0$.

as in fig. 2.3 a). For each symmetry $g$ we have a new category $\mathcal{C}_g$ which has charges $a_g$. The number of distinct $a_g$ is equal to the number of $g$-invariant charges in $\mathcal{C}_0$. When fusing charges $a_g$ and $b_h$ we must have that $a_g \times b_h = c_{g\cdot h}$ where $g \cdot h$ is the composition of elements of $G$. This is called $G$-graded fusion.

Since $g \cdot 0 = g$ we have that $a_g \times b_0 = a'_g$. If the $g$-invariant charge(s) in $\mathcal{C}_0$ cannot distinguish $b_0$ from the vacuum then $a'_g = a_g$ and we say that anyon $b_0$ is "localised" by twist $a_g$. Such charges and twists have fusion/splitting rules such that

$$N^{a_g}_{a_g b_0} = N^{a_g}_{a_g \bar{b}_0} = N^{b_0}_{a_g \bar{a}_g} = N^{\bar{a}_g}_{\bar{a}_g b_0} = N^{a_g}_{g b_0 a_g} \neq 0 \tag{2.12}$$

In other words if $a_g$ localises $b_0$:

- $\bar{a}_g$ also localises $b_0$.
- $b_0$ is one of the possible fusion outcomes of $a_g \times \bar{a}_g$.
- All charges in the orbit of $b_0$ under the action of $g$ are also localised by $a_g$ and $\bar{a}_g$.

Additionally we note that the set of localisable charges for a particular twist must be closed under fusion since if $a_0$ and $b_0$ both braid trivially with the $g$-invariant charges in $\mathcal{C}_0$ then so must the result of their fusion.

Braiding of charges $a_g$ and $b_h$ involves the action of the relevant symmetries such that charges can be modified by these braids (this is the G-crossed braiding part of the formalism). However, we will not require this part of the theory for reasons that will become clear below.

| | | |
|---|---|---|
| $rx$ | $gx$ | $bx$ |
| $ry$ | $gy$ | $by$ |
| $rz$ | $gz$ | $bz$ |

**Table 2.1:** The nine non-trivial bosonic anyons of the 2D colour code arranged as in [Kesselring et al., 2018]. $rx$ implies an $X$ error on a red plaquette and so on.

Finally, we note that fusion rules in this formalism must still satisfy the pentagon equation. Braiding rules are generalised to follow a "heptagon equation" which accounts for the fact that braiding with twists can alter charge labels.

We return now to the previously discussed case of twists in the toric code. $\mathcal{C}_0$ in this case is the quantum double of $\mathbb{Z}_2$ which has only one symmetry so $G$ has only two elements, 0 and $g$, where 0 is the identity. $\mathcal{C}_0$ contains two $g$-invariant charges ($\mathbf{1}$ and $\varepsilon$) so $\mathcal{C}_g$ has two charges which can be distinguished by braiding with $\varepsilon$. More specifically there is one charge corresponding to the fusion of the twist with either $\mathbf{1}$ or $\varepsilon$ and one charge corresponding to fusion with $e$ or $m$. This twist possesses two significant features: (1) it is self-inverse and (2) its associated invariant charges are also its localisable charges. This means that the subset of charges consisting of this twist and its localisable charges is closed under fusion. Furthermore, none of the charges in this subset can be altered by braiding with any of the others. In other words this subset functions as an anyon model - specifically the Ising anyon model. This is precisely what was noticed by H. Bombín in [Bombín, 2010], although the argument in that paper was formulated in terms of topological string operators.

In the 2D colour code the situation is not quite so simple. In [Kesselring et al., 2018] the 72 twists of the colour code are identified and arranged into nine conjugacy classes. The authors of [Kesselring et al., 2018] point out that the action of these twists can best be understood by considering the nine (non-trivial) bosonic anyons of the colour code arranged as in table 2.1. These anyons are all self-inverse and Abelian. Two anyons in a row or column fuse to the third and braid trivially with each other. Two anyons which do not share a row or column fuse to a fermion and aquire a phase of -1 under full exchange (monodromy).

The symmetries of the colour code anyon model are the permutations of this table which preserve the rows and columns. These permutations are the column permutations (6 options), row permutations (6 options) and the transpose (2 options) giving $6 \times 6 \times 2 = 72$ possible symmetries. Twists belonging to three of the nine conjugacy classes possess the same properties as the surface code twist described above: they are self-inverse and their associated sets of invariant and localisable charges are equivalent. One of the other six classes is trivial (it contains only the identity twist) and twists in the other five classes possess neither of these properties.

In what follows we consider the general case of the anyon model associated with these self-inverse twists and their localisable charges. In section 2.6 we will show that twists in any number of stacked surface codes can only have an invariant set of localisable charges if they are self-inverse.

## 2.2 A Hierarchy of Models

Recall the standard Ising model in which we have a single non-Abelian anyon $\sigma$ and two Abelian anyons $\mathbf{1}$ and $\psi$ such that $\sigma \times \sigma = \mathbf{1} + \psi$. We can extend this model by including additional Abelian anyons and modifying the outcome of $\sigma \times \sigma$ to include these anyons. Such extended models already exist in the literature in the form of parafermions, for example in [Hutter et al., 2015], but the Abelian anyons in these models are not generally self-inverse and so cannot be the natural anyons of the colour code. If we write the Abelian anyons of an extended Ising model as $\alpha_i$ (where $\alpha_0 = \mathbf{1}$) and the non-Abelian anyon as $\beta$ and require that each $\alpha_i$ must be its own antiparticle then we obtain the following fusion relations

$$\alpha_i \times \alpha_j = \alpha_k, \;\; \alpha_i \times \beta = \beta, \;\; \beta \times \beta = \sum_{i=0}^{n} \alpha_i \tag{2.13}$$

where $k = 0$ only if $i = j$ and $k = i$ ($k = j$) only if $j = 0$ ($i = 0$). These are exactly the fusion relations we observe for self-inverse twists and their localisable anyons in the colour code. For example, the colour code twist $B$ is self-inverse and can localise the anyons $bx, by, bz$ from table 2.1 (as well as the vacuum anyon). The set $\{\mathbf{1}, bx, by, bz\}$ is closed under fusion and so if we write $\mathbf{1} = \alpha_0$, $bx = \alpha_1$, $by = \alpha_2$,

$bz = \alpha_3$ and $B = \beta$ then fusion relations of these five charges exactly match (2.13).

Only specific values of $n$ yield valid extended Ising models. For example, the model $\{\alpha_0, \alpha_1, \alpha_2, \beta\}$ is not valid because it is not closed under fusion ($\alpha_1 \times \alpha_2$ must have a fusion outcome $\alpha_k$ where $k \neq 0, 1, 2$). Given a valid extended Ising model containing $m$ $\alpha$s we can find the next valid model with $n > m$ by adding a single new charge $\alpha_{m+1}$ to the model, fusing $\alpha_{m+1}$ with all existing charges, and adding all fusion outcomes to the model. If we write these fusion outcomes as $\alpha_i \times \alpha_{m+1} = \alpha_{i+m+1}$ then we can see that the resulting model must be closed under fusion since

- Fusion of any $\alpha_i, \alpha_j$ with $i, j \leq m$ results in another $\alpha_k$ with $k \leq m$ since the initial model was closed under fusion.

- Fusion of $\alpha_{m+1}$ with any anyon in the model results in another anyon in the model due to the above procedure.

- Fusion of any $\alpha_i, \alpha_j$ with $i \leq m$ and $j \geq m+2$ can be written as $\alpha_{i \leq m} \times \alpha_{j \geq m+2} = \alpha_{i \leq m} \times \alpha_{k \leq m} \times \alpha_{m+1}$ by definition of $\alpha_{j \geq m+2}$, and this is in the model due to the above two points.

- Fusion of any $\alpha_i, \alpha_j$ with $i, j \geq m+2$ can be written as $\alpha_{i \geq m+2} \times \alpha_{j \geq m+2} = \alpha_{k \leq m} \times \alpha_{m+1} \times \alpha_{l \leq m} \times \alpha_{m+1}$ which is in the model since the two $\alpha_{m+1}$s cancel.

Thus we can inductively define all extended Ising models beginning from the standard Ising model: $\{\alpha_0, \alpha_1, \beta\}$. We can label these models by $I_k$ where $k = 1$ is the standard Ising model. The number of $\alpha_i$ in a given $I_k$ is $n_k = 2n_{k-1} = 2^{k-1} n_1$ and $n_1 = 2$ so $n_k = 2^k$. The $\beta$ anyon for each $I_k$ can be written as $\beta_k$, and it has quantum dimension $\sqrt{2^k}$.

Note that we can equivalently define these models from multiple copies of $\mathbb{Z}_2$. The Abelian charges of the standard Ising anyon model form a group (with composition of group elements described by the model's fusion rules) that is isomorphic to $\mathbb{Z}_2$. Similarly the Abelian charges of $I_2$ form a group that is isomorphic to $\mathbb{Z}_2 \times \mathbb{Z}_2$ and so on. In general the group of Abelian charges of $I_k$ will be isomorphic to $k$ copies

of $\mathbb{Z}_2$. A set consisting of a finite group $A$ and a single additional element $\beta$ with composition of these elements defined as in (2.13) is called a Tambara-Yamagami (TY) category [Tambara and Yamagami, 1998]. Thus our "extended Ising hierarchy" can be described more formally as a hierarchy of TY categories with categories in the $k^{\text{th}}$ level of the hierarchy having base group $(\mathbb{Z}_2)^k$.

## 2.3 F Matrices

We now derive the possible F matrices $F^{\beta}_{\beta\beta\beta}$ for general $\beta_k$. These matrices (up to symmetry-preserving row and column permutations) are found to be a subset of the Hadamard matrices called Sylvester or Walsh matrices [Sylvester, 1867] multiplied by a constant. We then examine the trace of these matrices, as this will be relevant in the next section.

A general F-matrix for extended Ising anyons can be found from the pentagon equation (2.4). We will be using this equation extensively, so we restate it here for convenience.

$$(F^5_{12c})^d_a (F^5_{a34})^c_b = \sum_e (F^d_{234})^c_e (F^5_{1e4})^d_b (F^b_{123})^e_a \tag{2.14}$$

Recall that $(F^s_{pqr})^j_i$ describes a reordering of the fusion of three anyons $p$, $q$ and $r$. Prior to reordering we first fuse $p$ and $q$ to get $i$, then fuse $i$ with $r$ to get $s$. After reordering we fuse $q$ and $r$ to get $j$, then fuse $j$ with $p$ to get $s$. $(F^s_{pqr})^j_i$ is equal to zero if it describes a reordering that is disallowed by the fusion rules and equal to a phase or a matrix otherwise. $F^{\beta}_{\beta\beta\beta}$ is a matrix which we will henceforth refer to as $\boldsymbol{F}$. We let $\mathbf{1} = \alpha_0$ and write the elements of $\boldsymbol{F}$ as $F_{ij} = (F^{\beta}_{\beta\beta\beta})^{\alpha_j}_{\alpha_i}$. All other $(F^s_{pqr})^j_i$ are phases. We note some important properties of these phases and their inverses:

1. Every phase $(F^s_{pqr})^j_i$ has an inverse $(F^s_{rqp})^i_j$ since changes of fusion order are always reversible.

2. $(F^s_{pqr})^j_i$ where $p$,$q$ or $r = \mathbf{1}$ are trivial reorderings and correspond to a phase of 1.

3. Phases of the type $(F^s_{pqp})^i_i$ are self-inverse and so have value either 1 or $-1$.

We begin with the case where $1,2,3,4=\beta$, $5=\mathbf{1}$, $b=d=\beta$, $a=\alpha_i$ and $c=\alpha_j$. We obtain constraints

$$(F^{\mathbf{1}}_{\beta\beta\alpha_j})^{\beta}_{\alpha_i}(F^{\mathbf{1}}_{\alpha_i\beta\beta})^{\alpha_j}_{\beta}=\sum_{x=0}^{2^k-1}F_{xj}(F^{\mathbf{1}}_{\beta\alpha_x\beta})^{\beta}_{\beta}F_{ix} \tag{2.15}$$

In the case that $i=j$ the LHS of this equation is equal to 1 by property 1 as listed above. If $i\neq j$ the LHS is equal to 0 since $\alpha_i\times\alpha_j=\mathbf{1}$ is only possible when $i=j$. If we write $(F^{\mathbf{1}}_{\beta\alpha_x\beta})^{\beta}_{\beta}=\theta_x$ and sum over repeated indices then we can rewrite (2.15) as

$$F_{ix}F'_{xj}=\delta_{ij} \tag{2.16}$$

where each element $F'_{ij}=\theta_i\cdot F_{ij}$ (we do not sum over the repeated index here). The LHS of (2.16) is the matrix $\boldsymbol{F}$ and its inverse. $\boldsymbol{F}$ is unitary so we must have that

$$F_{ij}=(F'_{ij})^{\dagger}. \tag{2.17}$$

By property 3 $\theta_i=\pm1$ and by property 2 $\theta_0=1$. Thus

$$F_{i0}=(F'_{0i})^{*}=(\theta_0\cdot F_{0i})^{*}=(F_{0i})^{*} \tag{2.18}$$

and

$$F_{0j}=(F'_{j0})^{*}=(\theta_j\cdot F_{j0})^{*}=(\pm F_{j0})^{*} \tag{2.19}$$

and we can combine these to show

$$F_{0x}=(\pm F_{x0})^{*}=(\pm(F_{0x})^{*})^{*}=\pm F_{0x}. \tag{2.20}$$

Thus if $\theta_x=-1$ we must have $F_{0x}=F_{x0}=0$ for all $x\geq 0$.

The next configurations of the pentagon equation that we consider are $1=\alpha_x$, $2,3,4=\beta$ and $2=\alpha_x$, $1,3,4=\beta$. The first of these yields constraints

$$F_{ij}=\frac{(F^{\beta}_{\alpha_x(\alpha_i\times\alpha_x)\beta})^{\beta}_{\alpha_i}(F^{\alpha_i}_{\alpha_x\beta\beta)})^{\alpha_i\times\alpha_x}_{\beta}}{(F^{\beta}_{\alpha_x\beta\alpha_j})^{\beta}_{\beta}}F_{(i\times x)j} \tag{2.21}$$

where $F_{(i\times x)j} = (F^{\beta}_{\beta\beta\beta})^{\alpha_j}_{\alpha_i\times\alpha_x}$. The second configuration yields

$$F_{ij} = \frac{(F^{\alpha_j\times\alpha_x}_{\alpha_x\beta\beta})^{\alpha_j}_{\beta}(F^{\alpha_i}_{\beta\alpha_x\beta})^{\beta}_{\beta}}{(F^{\beta}_{\beta\alpha_x\alpha_j})^{\alpha_j\times\alpha_x}_{\beta}}F_{i(j\times x)} \tag{2.22}$$

From (2.21) we have that

$$F_{00} = (F^{\beta}_{\alpha_x\alpha_x\beta})^{\beta}_{\mathbf{1}}(F^{\mathbf{1}}_{\alpha_x\beta\beta})^{\alpha_x}_{\beta}F_{x0} \tag{2.23}$$

and from (2.22)

$$F_{i0} = (F^{\alpha_{x'}}_{\alpha_{x'}\beta\beta})^{\mathbf{1}}_{\beta}(F^{\alpha_i}_{\beta\alpha_{x'}\beta})^{\beta}_{\beta}F_{ix'} \tag{2.24}$$

Setting $x = i$ and $x' = j$ we can combine these equations to obtain

$$F_{ij} = (F^{\beta}_{\alpha_j\alpha_j\beta})^{\beta}_{\mathbf{1}}(F^{\mathbf{1}}_{\alpha_j\beta\beta})^{\alpha_j}_{\beta}(F^{\alpha_i}_{\alpha_i\beta\beta})^{\mathbf{1}}_{\beta}(F^{\alpha_j}_{\beta\alpha_i\beta})^{\beta}_{\beta}F_{00} \tag{2.25}$$

Thus all elements $F_{ij}$ have magnitude equivalent to that of $F_{00}$ and so the only way for $\theta_x = -1$ is to have $F_{00} = 0$ but this contradicts the constraint that $F_{ix}F'_{xi} = 1$ since all the terms in this sum would be 0. Thus all $\theta_x = 1$ and $\boldsymbol{F}$ must be Hermitian. Additionally, the magnitude of all elements in the matrix must be $1/\sqrt{2^k}$ and since the diagonal elements must be real $F_{00} = \pm 1/\sqrt{2^k}$.

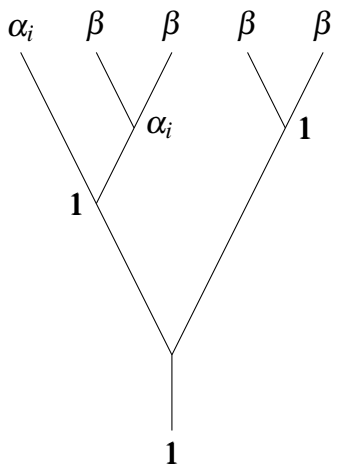
Setting

$$\begin{aligned} \phi_{ij} &= (F^{\alpha_j}_{\beta\alpha_i\beta})^{\beta}_{\beta}, \\ f_{i0} &= (F^{\alpha_i}_{\alpha_i\beta\beta})^{\mathbf{1}}_{\beta}, \\ f_{0j} &= (F^{\beta}_{\alpha_j\alpha_j\beta})^{\beta}_{\mathbf{1}}(F^{\mathbf{1}}_{\alpha_j\beta\beta})^{\alpha_j}_{\beta} \end{aligned} \tag{2.26}$$

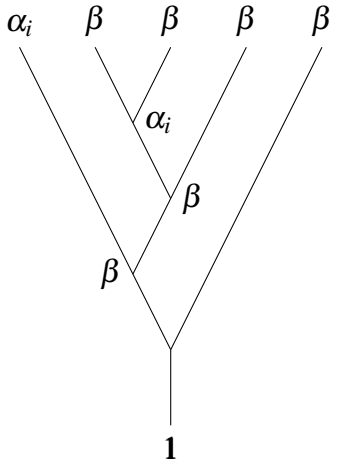
we can rewrite (2.25) as

$$F_{ij} = \pm\frac{\phi_{ij}f_{i0}f_{0j}}{\sqrt{2^k}}. \tag{2.27}$$

By property 3 all $\phi_{ij} = \pm 1$ (with $\phi_{i0} = \phi_{0j} = 1$ by property 2) while $f_{i0}$ can be any phase. We also have that $f_{0j} = (f_{j0})^*$ which can be verified by considering the five-anyon fusion tree




and observing that the F-move sequences $(F^{\beta}_{\alpha_i\alpha_i\beta})^{\beta}_{\mathbf{1}}(F^{\mathbf{1}}_{\beta\beta\mathbf{1}})^{\beta}_{\mathbf{1}}$ and $(F^{\mathbf{1}}_{\beta\beta\alpha_i})^{\beta}_{\alpha_i}(F^{\alpha_i}_{\beta\beta\alpha_i})^{\beta}_{\mathbf{1}}(F^{\mathbf{1}}_{\alpha_i\alpha_i\mathbf{1}})^{\alpha_i}_{\mathbf{1}}$ both map this tree to




.

Equating these and eliminating trivial terms using property 2 we get

$$(F^{\beta}_{\alpha_i\alpha_i\beta})^{\beta}_{\mathbf{1}} = (F^{\mathbf{1}}_{\beta\beta\alpha_i})^{\beta}_{\alpha_i}(F^{\alpha_i}_{\beta\beta\alpha_i})^{\beta}_{\mathbf{1}} \tag{2.28}$$

which can be rewritten as

$$(F^{\beta}_{\alpha_i\alpha_i\beta})^{\beta}_{\mathbf{1}}(F^{\mathbf{1}}_{\alpha_i\beta\beta})^{\alpha_i}_{\beta} = ((F^{\alpha_i}_{\alpha_i\beta\beta})^{\mathbf{1}}_{\beta})^{-1} \tag{2.29}$$

using property 1.

We can then write $\boldsymbol{F}$ as the Hadamard product of a matrix of $f$s ($\boldsymbol{f}$) and a matrix

of $\phi$s ($\boldsymbol{\phi}$) multiplied by $1/\sqrt{2^k}$

$$\boldsymbol{F} = \pm\frac{1}{\sqrt{2^k}}(\boldsymbol{\phi}\circ\boldsymbol{f}). \tag{2.30}$$

$\boldsymbol{f}$ is Hermitian and so $\boldsymbol{\phi}$ must also be Hermitian. The multiplication of $\boldsymbol{F}$ by itself gives

$$\begin{aligned} F_{ix}F_{xj} &= \frac{\phi_{ix}f_{i0}f_{0x}}{\sqrt{2^k}}\frac{\phi_{xj}f_{x0}f_{0j}}{\sqrt{2^k}} \\ &= \frac{1}{2^k}\cdot\phi_{ix}\phi_{xj}\cdot f_{i0}f_{0j} \\ &= \delta_{ij} \end{aligned} \tag{2.31}$$

where we have used the fact that $f_{0x}f_{x0} = 1$. The final equivalence implies

$$\frac{1}{2^k}\boldsymbol{\phi}^2\circ\boldsymbol{f} = \boldsymbol{I} \tag{2.32}$$

where $\boldsymbol{I}$ is the $2^k\times 2^k$ identity matrix. The diagonal elements of $\boldsymbol{f}$ are all 1 (since $f_{0i} = (f_{i0})^*$) so we must have that $\boldsymbol{\phi}^2 = 2^k\boldsymbol{I}$. Thus $\boldsymbol{\phi}$ is a symmetric Hadamard matrix (Hadamard matrices are $n\times n$ matrices where all entries are $\pm 1$ and $\boldsymbol{M}\boldsymbol{M}^T = n\boldsymbol{I}$ [Hedayat and Wallis, 1978]). Additionally, $\boldsymbol{\phi}$ as we have defined it has the property that all entries in the first row and column are $+1$ (such Hadamard matrices are called "normalised" but we will avoid using this term to prevent confusion with its more common usage in quantum physics).

$\boldsymbol{\phi}$ has only a finite number of solutions while $\boldsymbol{f}$ has an infinite number. We infer from this that different $\boldsymbol{\phi}$ correspond to different anyon models (with the discreteness of these solutions consistent with Ocneanu rigidity [Kitaev, 2006]), while the different $\boldsymbol{f}$ correspond to a choice of gauge. We can see that we cannot transform between solutions of $\boldsymbol{\phi}$ by changes to $\boldsymbol{f}$ by observing that $\boldsymbol{f}$ is completely characterised by the values of $f_{i0}$, whereas $\phi_{i0}$ are always 1, so changes to $\boldsymbol{f}$ are always reflected in the first row of $F^{\beta}_{\beta\beta\beta}$ while changes to $\boldsymbol{\phi}$ are not. We also show in the next section that the braiding matrices of the models depend only on $\boldsymbol{\phi}$ and

not on $\boldsymbol{f}$. Thus we can make the gauge choice that all $\boldsymbol{f} = 1$ so

$$\boldsymbol{F} = \pm\frac{1}{\sqrt{2^k}}\boldsymbol{\phi}. \tag{2.33}$$

Note that instead of (2.23) and (2.24) we could have obtained

$$F_{0j} = (F^{\beta}_{\alpha_x\alpha_x\beta})^{\beta}_{\mathbf{1}}(F^{\mathbf{1}}_{\alpha_x\beta\beta})^{\alpha_x}_{\beta}(F^{\beta}_{\alpha_j\beta\alpha_x})^{\beta}_{\beta}F_{xj} \tag{2.34}$$

from (2.21) and

$$F_{00} = (F^{\alpha'_x}_{\alpha'_x\beta\beta})^{\mathbf{1}}_{\beta}F_{0x'} \tag{2.35}$$

from (2.22). Setting $x = i$ and $x' = j$ and combining these as before gives

$$F_{ij} = (F^{\mathbf{1}}_{\beta\beta\alpha_i})^{\beta}_{\alpha_i}(F^{\beta}_{\beta\alpha_i\alpha_i})^{\mathbf{1}}_{\beta}(F^{\alpha_j}_{\beta\beta\alpha_j})^{\beta}_{\mathbf{1}}(F^{\beta}_{\alpha_i\beta\alpha_j})^{\beta}_{\beta}F_{00}. \tag{2.36}$$

Using property 1 we can see that the first three terms are equal to $(f_{j0}f_{0i})^{-1}$ which is equal to 1 due to our choice of gauge. Thus we have that

$$\frac{1}{\sqrt{2^k}}\phi_{ij} = \frac{1}{\sqrt{2^k}}(F^{\beta}_{\alpha_i\beta\alpha_j})^{\beta}_{\beta} \tag{2.37}$$

so

$$\phi_{ij} = (F^{\alpha_j}_{\beta\alpha_i\beta})^{\beta}_{\beta} = (F^{\beta}_{\alpha_i\beta\alpha_j})^{\beta}_{\beta} \tag{2.38}$$

with this gauge choice.

We can further restrict $\boldsymbol{\phi}$ to a subset of Hadamard matrices by using the group theory and TY category connections discussed previously. The F matrices of TY categories are related to symmetric non-degenerate bicharacters where "character" refers to multiplicative character, i.e. a group homomorphism from a group $A$ to the multiplicative group of a field $\boldsymbol{F}^{\times}$ [Emil Artin, 1959]. A bicharacter is then a

function $\chi : A \times A \to \boldsymbol{F}^{\times}$ which satisfies [Rozanski, 1996]

$$\begin{aligned} &\chi(a_1 a_2, a_3) = \chi(a_1, a_3)\chi(a_2, a_3) \\ &\text{and} \\ &\chi(a_1, a_2 a_3) = \chi(a_1, a_2)\chi(a_1, a_3). \end{aligned} \tag{2.39}$$

We can see that $\boldsymbol{\phi}$ satisfies this by considering the pentagon equation with $1 = 3 = \beta$, $2 = \alpha_i$, $4 = \alpha_j$ and $5 = \alpha_k$ which yields a constraint

$$(F^{\alpha_k}_{\beta\alpha_i\beta})^{\beta}_{\beta} = (F^{\beta}_{\alpha_i\beta\alpha_j})^{\beta}_{\beta}(F^{(\alpha_j\times\alpha_k)}_{\beta\alpha_i\beta})^{\beta}_{\beta} \tag{2.40}$$

which can be rewritten as

$$\phi_{i(j\times k)} = \phi_{ij}\phi_{ik} \tag{2.41}$$

using (2.38) and the fact that $(\phi_{ij})^{-1} = \phi_{ij}$ since $\phi_{ij} = \pm 1$. $\boldsymbol{\phi}$ is symmetric and so we also have that

$$\phi_{(j\times k)i} = \phi_{ji}\phi_{ki}. \tag{2.42}$$

Thus $\boldsymbol{\phi}$ is a bicharacter on $(\mathbb{Z}_2)^k$. The above definition means that if we fix one argument of a bicharacter on a group $A$ then the bicharacter as a function of the other argument defines a character on $A$. In other words, each row and column of $\boldsymbol{\phi}$ is a character on $(\mathbb{Z}_2)^k$. Because they are homomorphisms the action of these characters on $(\mathbb{Z}_2)^k$ is defined by their action on each of the $k$ copies of $\mathbb{Z}_2$ which make it up. There are two valid $\pm 1$ valued characters on $\mathbb{Z}_2$ which are given by the rows/columns of the $2 \times 2$ Hadamard matrix

$$H_1 = \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \tag{2.43}$$

and coincide with the irreducible representations of $\mathbb{Z}_2$. Thus for $(\mathbb{Z}_2)^k$ there are $2^k$ possible characters corresponding to the rows/columns of $H_1^{\otimes k}$. These matrices are a subset of the Hadamard matrices called Sylvester or Walsh matrices [Sylvester, 1867]. The possible bicharacters for $(\mathbb{Z}_2)^k$ are then the Sylvester matrix $H_1^{\otimes k}$ and

any other matrices which can be obtained from this matrix via symmetry-preserving row/column permutations.

We now investigate the possible values of the trace of these matrices as this will be important when we consider $R$ matrices in the next section. $H_1$ is the unique bicharacter for $k = 1$ and has trace 0. For $k = 2$ there are four possible bicharacters which we can write in matrix form as

$$\begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & 1 & -1 \\ 1 & -1 & -1 & 1 \end{pmatrix} \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \end{pmatrix} \\ \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & -1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 \end{pmatrix} \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \\ 1 & -1 & 1 & -1 \end{pmatrix}. \tag{2.44}$$

One of these matrices has trace 4 while the other three have trace 0. These three all correspond to the same anyon model up to relabelling of charges. Below, we show that symmetry-preserving permutations of columns of a symmetric Hadamard matrix must alter the trace of the matrix by either 0 or $\pm 2^k$, with the latter only being possible for even $k$. The Sylvester matrices all have trace 0 and since one row/column contains only $+1$s we cannot have $\mathrm{Tr}(\boldsymbol{\phi}) = -2^k$ so the only possible traces are 0 for odd $k$ and $0, 2^k$ for even $k$. In comparison general symmetric $2^k \times 2^k$ Hadamard matrices must also have trace 0 for odd $k$ but the trace can take any value $2^k \geq n2^{k/2+1} \geq 0$ for even $k$ and integer $n$ [Craigen, 1994].

**Theorem 2.1** *Column permutations that preserve the symmetry of a symmetric* $2^k \times 2^k$ *Hadamard matrix must alter the trace by either* 0 *or* $\pm 2^k$.

Consider swapping the second and third columns of the following matrix from (2.44). This matrix corresponds to $H_1 \otimes H_1$ and this column swap will result in the matrix from (2.44) with trace 4.

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

We have divided this pair of columns into three sections. Any changes to the top or bottom (off-diagonal) sections will result in a non-symmetric matrix but the central (diagonal) section can be modified while preserving symmetry. Because these two columns are identical in the off-diagonal sections and the diagonal section is symmetric both before and after the exchange the symmetry of the overall matrix is preserved. However, for $k > 2$ such exchanges cannot be symmetry preserving because the diagonal section will always contain 2 elements from each column while the off-diagonal sections will contain the other $2^k - 4$. In order for the columns of the matrix to be orthogonal each pair of columns must match in exactly half their entries so for matrices larger than $4 \times 4$ it is impossible to exchange two columns in such a way that the off-diagonal sections are unchanged. Instead, we must exchange sets of columns. Consider a general $2^k \times 2^k$ matrix broken into $2^{k-2} \times 2^{k-2}$ blocks as follows

$$\begin{pmatrix} E & F \\ A & B \\ C & D \\ G & H \end{pmatrix}$$

where we have once again marked diagonal and off-diagonal sections.

**Lemma 2.1.1** *The exchange of EACG and FBDH can only preserve the symmetry of the matrix if the trace of the diagonal section is negated by the exchange.*

We can only exchange $EACG$ and $FBDH$ if $E = F$ and $G = H$. Additionally we must have that $A = -B$ and $C = -D$ or the columns of the matrix would not be orthogonal. Thus the trace of the diagonal section is negated by this exchange. □

**Lemma 2.1.2** *The exchange of EACG and FBDH can only preserve the symmetry of the matrix if A is a symmetric Hadamard matrix*

For the overall matrix to be symmetric both before and after the exchange we require that $B^T = C$ and $A^T = D$ and therefore $A = -B = -C^T = D^T$ and the entire diagonal section is determined by $A$.

$A$ must be symmetric for the overall matrix to be symmetric. To show that the columns of $A$ are orthogonal (so that $AA^T = 2^{k-2}\boldsymbol{I}$) we consider two columns, $c_1$ and $c_2$, of our full $2^k \times 2^k$ matrix such that both columns pass through $A$. We say that two elements $c_{1,i}$ and $c_{2,i}$ "match" if $c_{1,i} = c_{2,i}$. If two elements do not match then $c_{1,i} = -c_{2,i}$. We note that if this pair of columns have $m$ matching elements within $A$ then they must have $2m$ matching elements within the diagonal section and $2^{k-1} - 2m$ matching elements in the off-diagonal sections (since exactly half the elements of each column must match). We now consider the pair $c_1$ and $-c_2$ where $-c_2$ is the column passing through $B$ that is equal to $-1 \times c_2$ within the diagonal section and identical to $c_2$ outside of this section (i.e. the column we wish to exchange with $c_2$). $c_1$ and $-c_2$ have $2^{k-1} - 2m$ matching elements within the diagonal section and so must have $2m$ matching elements in the off-diagonal sections. Since $c_2$ and $-c_2$ are identical in the off-diagonal sections we have that $2m = 2^{k-1} - 2m$ and $m = 2^{k-3}$. Thus for each pair of columns in $A$ half of the elements match and the other half do not and the columns of $A$ are orthogonal. □

Since $A$ is a symmetric Hadamard matrix of order $2^{k-2}$ it can be partitioned into blocks such that $A'$ is a symmetric Hadamard matrix of order $2^{k-4}$. This process can be repeated recursively, eventually terminating when we arrive at a matrix of order 1 (for odd $k$) or order 2 (for even $k$). The possible symmetric Hadamard matrices with these orders are (up to a possible global phase of -1) (2.43) and (2.44). These matrices are restricted to have trace=0 and trace=0 or 4 respectively, and going back up the chain of recursion we see that $\mathrm{Tr}(A) = 0$ for odd $k$ and $\mathrm{Tr}(A) = 0, \pm 2^{k-2}$ for even $k$. The trace of the central section is then $2\mathrm{Tr}(A)$ and by lemma 2.1.1 a symmetry preserving exchange of columns must negate this trace, changing the trace of the overall matrix by either 0 for odd $k$ or $0, 2^k$ for even $k$.

So far we have only considered swapping columns in a very specific arrangement, but any desired set of column swaps can be rewritten in this form such that it

is apparent that the same constraints apply. This is achieved by applying column permutations such that the columns we wish to swap are correctly arranged at the centre of the matrix and then applying a matching set of row permutations (since matching column and row permutations preserve the symmetry of the matrix). Following the exchange of column blocks as described above we apply the same set of row and column permutations again. If we write an exchange of the $i^{\text{th}}$ and $j^{\text{th}}$ columns as $P_{ij}$ and an exchange of rows as $Q_{ij}$ then, for the exchange of columns $a$ and $b$, the sequence of operations described above is

$$Q_{bd}Q_{ac}P_{bd}P_{ac}P_{cd}Q_{bd}Q_{ac}P_{bd}P_{ac} = P_{bd}P_{ac}P_{cd}P_{ac}P_{bd}. \tag{2.45}$$

because row permutations commute with column permutations and row (column) permutations commute with other row (column) permutations if they act on separate pairs of columns. Also, $P_{ij}P_{ik}P_{ij} = P_{jk}$ and so

$$P_{bd}P_{ac}P_{cd}P_{ac}P_{bd} = P_{bd}P_{ad}P_{bd} = P_{ab} \tag{2.46}$$

and so this operation is equivalent to exchanging the columns without first moving them to the center. □

## 2.4 R Matrices

We now discuss the possible R matrices for extended Ising models which are found from the hexagon equation (2.6). This equation was solved for the case of TY categories in [Siehler, 2000]. Here we present a similar solution in the language of section 2.1. As with the pentagon equation in the previous section we restate the hexagon equation here for convenience.

$$R^c_{13}(F^4_{213})^c_a R^a_{12} = \sum_b (F^4_{231})^c_b R^4_{1b}(F^4_{123})^b_a \tag{2.47}$$

We are concerned only with $R_{\beta\beta}$ which are diagonal matrices with elements $R^{\alpha_i}_{\beta\beta}$ indexed by $i$.

**Theorem 2.2** *The elements of $R_{\beta\beta}$ are given by*

$$R^{\alpha_i}_{\beta\beta} = \pm\sqrt{\phi_{ii}}R^{\alpha_0}_{\beta\beta} \tag{2.48}$$

*where $R^{\alpha_0}_{\beta\beta}$ has possible values $\{\pm 1, \pm i, \pm e^{i\pi/4}, \pm e^{-i\pi/4}\}$ for even $k$ or $\{\pm e^{i\pi/8}, \pm e^{-i\pi/8}, \pm ie^{i\pi/8}, \pm ie^{-i\pi/8}\}$ for odd k and the choices of $\pm$ for each element are consistent with table 2.2.*

We begin by setting $1,2,3,4 = \beta,\ a = \alpha_i,\ c = \alpha_j$ which gives:

$$R^{\alpha_j}_{\beta\beta}F_{ij}R^{\alpha_i}_{\beta\beta} = \sum_{b=0}^{2^k} F_{bj}R^{\beta}_{\beta\alpha_b}F_{ib} \tag{2.49}$$

and from (2.27)

$$\pm\frac{\phi_{ij}f_{i0}f_{0j}}{\sqrt{2^k}}R^{\alpha_i}_{\beta\beta}R^{\alpha_j}_{\beta\beta} = \sum_{b=0}^{2^k}\frac{\phi_{bj}f_{b0}f_{0j}\phi_{ib}f_{i0}f_{0b}R^{\beta}_{\beta\alpha_b}}{2^k} \tag{2.50}$$

$$\pm\frac{\phi_{ij}R^{\alpha_i}_{\beta\beta}R^{\alpha_j}_{\beta\beta}}{\sqrt{2^k}} = \sum_{b=0}^{2^k}\frac{\phi_{ib}\phi_{bj}R^{\beta}_{\beta\alpha_b}}{2^k} \tag{2.51}$$

Setting $1 = \alpha_i$ and $4 = \alpha_j$ we instead obtain

$$\phi_{ij}(R^{\beta}_{\alpha_i\beta})^2 = R^{\alpha_j}_{\alpha_i(\alpha_i\times\alpha_j)} \tag{2.52}$$

So we see that the braiding relations are dependent on $\boldsymbol{\phi}$ but not on $\mathbf{f}$ as we would expect.

Consider the case in (2.51) where $i = j$. We have that

$$\frac{\phi_{ii}(R^{\alpha_i}_{\beta\beta})^2}{\sqrt{2^k}} = \pm\sum_b\frac{R^{\beta}_{\beta\alpha_b}}{2^k}. \tag{2.53}$$

The RHS is independent of $i$ so

$$(R^{\alpha_0}_{\beta\beta})^2 = \phi_{ii}(R^{\alpha_i}_{\beta\beta})^2. \tag{2.54}$$

which we can rewrite as

$$R^{\alpha_i}_{\beta\beta} = \pm\sqrt{\phi_{ii}}R^{\alpha_0}_{\beta\beta}. \tag{2.55}$$

This proves the first part of theorem 2.2 and we now only need to fix the value of $R^{\alpha_0}_{\beta\beta}$ and the choice of $\pm$.

Returning to (2.51), if we set $i = 0$ and sum over $j$

$$\pm\sum_j \frac{R^{\alpha_0}_{\beta\beta}R^{\alpha_j}_{\beta\beta}}{\sqrt{2^k}} = \sum_j\sum_b \frac{\phi_{bj}R^{\beta}_{\beta\alpha_b}}{2^k} \tag{2.56}$$

all terms on the RHS cancel except for those where $b = 0$ which sum to $2^k R^{\beta}_{\beta\alpha_0}$ (since all rows/columns of $\boldsymbol{\phi}$ except for the first contain an equal number of 1s and $-1$s), so

$$\pm\sum_j \frac{R^{\alpha_0}_{\beta\beta}R^{\alpha_j}_{\beta\beta}}{\sqrt{2^k}} = R^{\beta}_{\beta\alpha_0} = 1 \tag{2.57}$$

since $R^{\beta}_{\beta\alpha_0}$ just describes braiding with the vacuum and is therefore trivial. Using (2.54) we can rewrite this as

$$\pm\frac{(R^{\alpha_0}_{\beta\beta})^2(1\pm\sqrt{\phi_{11}}\pm\ldots\pm\sqrt{\phi_{n_kn_k}})}{\sqrt{2^k}} = 1 \tag{2.58}$$

The $\pm$ in the sum do not all need to be the same, but they must be chosen such that $|R^{\alpha_0}_{\beta\beta}|^2 = 1$ or the matrix $R_{\beta\beta}$ would not be unitary.

The number of $+1$ ($-1$) terms in the diagonal of $\boldsymbol{\phi}$ tells us the number of $\pm1$ ($\pm i$) terms in the sum of (2.58) which we can rewrite as

$$\pm\frac{(R^{\alpha_0}_{\beta\beta})^2(a+ib)}{\sqrt{2^k}} = 1. \tag{2.59}$$

In order for $|R^{\alpha_0}_{\beta\beta}|^2 = 1$ we require that $a^2+b^2 = 2^k$. We can prove the following:

**Lemma 2.2.1** *The solutions to the equation $a^2+b^2 = 2^k$ are $a = \pm2^{k/2}, b = 0$ and $a = 0, b = \pm2^{k/2}$ for even $k$ and $a = \pm b = \pm2^{(k-1)/2}$ for odd $k$.*

**Proof:** Consider a right-angled triangle with sides $a \leq b < c$ opposed by angles

$A, B, C$.

- **Even:** $\sqrt{2^k} = 2^{k/2}$ is an integer so from $a^2 + b^2 = 2^k$ we assume the existence of a Pythagorean triple $(|a|, |b|, 2^{k/2})$. $a$ and $b$ must have the same parity for the sum of their squares to be even. If they are both odd then this triple is primitive since $|a|$ and $|b|$ have no even factors and $2^{k/2}$ has no odd factors. If they are both even then there must be some associated primitive triple $(|a|/2^x, |b|/2^x, 2^{k/2-x})$ where the first two elements are both odd. All primitive triples can be constructed using Euclid's formula

$$a = m^2 - n^2, \;\; b = 2mn, \;\; c = m^2 + n^2 \tag{2.60}$$

  where $m$ and $n$ are a pair of coprime integers, one of which is even. However, this means that $c$ is odd, giving a contradiction. Thus this primitive triple does not exist and neither does the triple $(|a|, |b|, 2^{k/2})$. The only remaining solutions to the equation $a^2 + b^2 = 2^k$ are $a = \pm 2^{k/2}, b = 0$ and $a = 0, b = \pm 2^{k/2}$

- **Odd:** $\sqrt{2^k} = 2^{k/2} = 2^{(k-1)/2}\sqrt{2}$ where $2^{(k-1)/2}$ is an integer power of two. Given $c = 2^{(k-1)/2}\sqrt{2}$ we have that $a = 2^{(k-1)/2}\sqrt{2}\sin(A)$ and $b = 2^{(k-1)/2}\sqrt{2}\sin(B)$. We require that both $a$ and $b$ are integers and thus $\sin(A)$ and $\sin(B)$ must both be integer multiples of $1/\sqrt{2}$. Thus $\sin(A) = \pm\sin(B) = \pm 1/\sqrt{2}$ and $a = \pm b = \pm 2^{(k-1)/2}$.

□

Substituting these solutions into (2.59) we have

$$\pm (R^{\alpha_0}_{\beta\beta})^2 = 1 \;\text{ and }\; \pm i(R^{\alpha_0}_{\beta\beta})^2 = 1 \tag{2.61}$$

for even $k$ and

$$\pm \frac{(R^{\alpha_0}_{\beta\beta})^2(1 \pm i)}{\sqrt{2}} = 1 \tag{2.62}$$

for odd $k$. Rearranging these we find that $R^{\alpha_0}_{\beta\beta}$ can take values $\{\pm 1, \pm i, \pm e^{i\pi/4}, \pm e^{-i\pi/4}\}$ and $\{\pm e^{i\pi/8}, \pm e^{-i\pi/8}, \pm ie^{i\pi/8}, \pm ie^{-i\pi/8}\}$ for even and odd $k$ respectively.

| $k$ | $\mathrm{Tr}(\boldsymbol{\phi})$ | Number of $\pm 1$ | Number of $\pm i$ |
|---|---|---|---|
| Even | $2^k$ | $2^{k-1} \pm 2^{k/2-1}$ | 0 |
| Even | 0 | $2^{k-2} \pm 2^{k/2-1}$ | $2^{k-2}$ |
| Odd | 0 | $2^{k-2} \pm 2^{(k-3)/2}$ | $2^{k-2} \pm 2^{(k-3)/2}$ |

**Table 2.2:** The possible numbers of $\pm 1$ and $\pm i$ on the diagonal of $R_{\beta\beta}$ up to global phase. The $\pm$ signs in a given column correspond to the $\pm$ in that column's header such that choosing the sign in the header to be $+$ also fixes all $\pm$ in the column to $+$.

Finally we consider the possible choices of $\pm$ in (2.55). We have actually solved this problem already, because we know the possible values of $\mathrm{Tr}(\boldsymbol{\phi})$ (which tell us how many $+1$s and $-1$s are on the diagonal of $\phi$ and thus how many $\pm 1$s and $\pm i$s are on the diagonal of $R_{\beta\beta}$) and we know the possible solutions to $a^2 + b^2 = 2^k$ (which tell us how many $+1$s ($+i$s) there should be relative to $-1$s ($-i$s)). These numbers are as listed in table 2.2. For instance, if $\mathrm{Tr}(\boldsymbol{\phi}) = 2^k$ then all $\phi_{ii} = 1$ and if we choose $R^{\alpha_0}_{\beta\beta} = 1$ then all diagonal elements of $R_{\beta\beta}$ are $\pm 1$. We then must have $a = \pm 2^{k/2}$ and $b = 0$, and if we choose $a = 2^{k/2}$ then we have $2^{k-1} + 2^{k/2-1}$ positive elements and $2^{k-1} - 2^{k/2-1}$ negative elements. Notice that there are several choices of global phase involved here (choice of $R^{\alpha_0}_{\beta\beta}$, choice of sign of $a$ etc) and so the numbers in table 2.2 are correct only up to such global phases. □

The elements $\phi_{ii}$ are also significant in another way. By setting $i = j$ in (2.52) we can show that

$$R^{\beta_k}_{\alpha_i\beta_k} = \pm\sqrt{\phi_{ii}}\sqrt{R^{\alpha_i}_{\alpha_i\mathbf{1}}} = \pm\sqrt{\phi_{ii}} \tag{2.63}$$

where $R^{\alpha_i}_{\alpha_i\mathbf{1}} = 1$ since braiding with the vacuum is trivial. Using this and instead setting $j = 0$ we find

$$(R^{\beta_k}_{\alpha_i\beta_k})^2 = \phi_{ii} = R^{\mathbf{1}}_{\alpha_i\alpha_i}. \tag{2.64}$$

In other words the elements $\phi_{ii}$ tell us the self-exchange statistics of the charges $\alpha_i$, with $\phi_{ii} = 1$ indicating that $\alpha_i$ is bosonic and $\phi_{ii} = -1$ telling us that $\alpha_i$ is fermionic. Thus we expect that for both odd and even $k$ we can obtain models containing $2^{k-1}$ bosonic and $2^{k-1}$ fermionic Abelian charges and a single non-Abelian charge. We additionally expect that for even $k$ we can obtain models containing $2^k$ bosonic Abelian charges, no fermionic charges and a single non-

Abelian charge. The models with $2^{k-1}$ bosonic and $2^{k-1}$ fermionic charges can be viewed as "sub-models" of $k$ copies of the standard Ising model (e.g. in some kind of multi-layer system) containing all Abelian charges and only a single non-Abelian charge (namely, the charge corresponding to $\sigma_1 \otimes \sigma_2 ... \otimes \sigma_k$). We note also that this "sub-model" simply corresponds to the case where we neglect some of the charges in the original model and does not mean that these charges are no longer present. It is therefore different from procedures such as that of Bais and Slingerland [Bais and Slingerland, 2009] in which an actual change to the model is made.

The models containing $2^k$ bosonic charges cannot be produced from copies of the standard Ising model and instead correspond to copies of a different anyonic model with four bosonic Abelian charges and a single non-Abelian charge.

## 2.5 Logical Gates

In this section we will rephrase the findings from the past two sections in terms of the possible logical gates which we can perform using these anyons.

Consider a specific anyon model containing $2^k$ Abelian charges and a single non-Abelian charge. This model has an $F$ matrix $F^{\beta}_{\beta\beta\beta}$ associated with changing the fusion order of three of the non-Abelian anyons and an $R$ matrix $R_{\beta\beta}$ associated with braiding two of the non-Abelian anyons. A system containing four such anyons has a $2^k$ dimensional fusion space for which the $2^k$ Abelian anyons of the model form a canonical basis. The $F$ and $R$ matrices provide us with two logical operations which can be performed on this space. The $F$ matrix is a mapping between the canonical basis and a basis of equal superpositions of the canonical basis vectors. The $R$ matrix selectively applies one of the phases $\{+1,-1,+i,-i\}$ to each vector of the canonical basis with the total number of $+1$s, $-1$s, $+i$s and $-i$s applied consistent with table 2.2.

Both of these operations may be interpreted in terms of Clifford gates on $k$ qubits: the $F$ matrix has the same rows and columns as a tensor product of $k$ Hadamard gates (up to a global phase) and the same is true for the $R$ matrices and tensor products of either $k$ S gates or $k/2$ CZ gates. Notice that our canonical basis

vectors are currently labelled only by anyonic charges and we have not yet defined an encoding for qubits in this space. We can always choose this encoding such that the ordering of the diagonal elements of $R$ in the computational basis is consistent with the respective tensor product of Clifford gates. The same is also true for the trace-0 $F$ matrices but not for the trace-$\sqrt{2^k}$ $F$ matrices since the trace is independent of our choice of encoding. These matrices cannot be decomposed into a tensor product of single qubit gates and instead correspond to tensor products of the trace-4 matrix in (2.44) multiplied by $1/2$. This matrix is equivalent to SWAP$\cdot(H \otimes H)$ and so is also Clifford. Thus, up to global phases and a choice of encoding, all $F$ and $R$ matrices implement Clifford operations on our Hilbert space.

## 2.6 Stacked Surface Codes

Part of our motivation for obtaining the results presented in the previous sections was to examine the braiding relations of twists in the 2D colour code. In this section we will see that we can obtain twists belonging to the first and second levels of the hierarchy in this code and in general we can obtain twists belonging to the $k^{\text{th}}$ level of the hierarchy in a stack of $k$ 2D surface codes. When visualising such a stack we might imagine multiple copies of the surface code placed one above the other, but we allow operations between any two layers regardless of how "far apart" in the stack they are and thus there is no notion of distance in a third dimension and the stack is still 2D. Equivalently, we can imagine a single 2D lattice with multiple qubits placed at each site.

The fact that models from the second level of the hierarchy can be realised in the 2D colour code is readily apparent from [Kesselring et al., 2018]. For example the twist $B$ which exchanges the $r$ and $g$ columns of table 2.1 has four nontrivial localisable charges: $\mathbf{1}$, $bx$, $by$ and $bz$. This set of four charges is closed under fusion and all four are invariant under the action of $B$.

In general if we consider equivalence up to global phases and choose our qubit

encoding as discussed in the previous section then there are two R-matrices for $k = 2$:

$$\begin{pmatrix} 1 & & & \\ & 1 & & \\ & & 1 & \\ & & & -1 \end{pmatrix} \quad \begin{pmatrix} 1 & & & \\ & i & & \\ & & i & \\ & & & -1 \end{pmatrix} \tag{2.65}$$

Recall that the first of these matrices belongs to a model with four bosonic Abelian charges while the second belongs to a model with two bosonic and two fermionic charges. In the notation of Kesselring et al [Kesselring et al., 2018] the models containing four bosons are those associated with a twist from conjugacy class B while those containing two bosons and two fermions are associated with twists from conjugacy class C. The twists in conjugacy class G have only two localisable charges and correspond to models from the first level of the hierarchy.

So far we have seen that we can realise the $k = 1$ level of the extended Ising hierarchy in the surface code, and the $k = 2$ level in the colour code, which is equivalent to two copies of the surface code [Kubica et al., 2015]. We now consider the general case of a stack of $k$ copies of the surface code.

Consider the anyon model of $k$ stacked surface codes (in the absence of twists). The topological charges in this model are the elements of a finitely-generated free group, whose generating set can be written $\{e_1, m_1, e_2, m_2, ..., e_k, m_k\}$ where the subscript shows the layer in the stack which the charge belongs to. Twists in the code stack correspond to symmetries of the anyon model. These symmetries can be formally defined as the elements of the automorphism group of the anyon model which preserve braiding relations. The action of these symmetries can be described via a set of orbits, each of which can be written as

$$a \to b \to c \to ... \to a \tag{2.66}$$

with the "trivial orbit" defined as

$$a \to a. \tag{2.67}$$

We first show that only self-inverse symmetries $g$ of this anyon model can have a $g$-invariant set of localisable charges. Consider a twist $t_g$ and a charge $b_0 = a_0 \times {}^g a_0$. $b_0$ can be localised by $t_g$ because we can split it into $a_0$ and ${}^g a_0$, then braid $a_0$ around $t_g$ and fuse it to the vacuum with ${}^g a_0$. If we braid $b_0$ around $t_g$ we obtain ${}^g b_0 = {}^g a_0 \times {}^{g^2} a_0$ and so in order for $b_0$ to be $g$-invariant we must have ${}^{g^2} a_0 = a_0$, implying that $g$ is self-inverse.

From this we can see that if a twist in a stack of surface codes (together with its set of localisable charges) can be considered as an anyon model this model will belong to the hierarchy of extended Ising models defined in section 2.2.

All non-trivial orbits associated with a self-inverse symmetry have the form $a \to b \to a$ which can also be written $a \leftrightarrow b$.

The full automorphism group of a finitely generated free group with ordered basis $[x_1, ..., x_n]$ can be generated by the *elementary Neilsen transformations*[Magnus et al., 2004]:

- Switch $x_1$ and $x_2$
- Replace $x_1$ with $x_1^{-1}$
- Replace $x_1$ with $x_1 \cdot x_2$

The second transformation is equal to the identity transformation in our case because all charges in our model are their own inverse. We thus consider only the first and third transformations, but not all applications of these transformations are valid because we must also preserve braiding relations. In order to do this we require that if we map $x_i \to x_j$ then we must also map $x'_i \to x'_j$ and if we map $x_i \to x_i x_j$ then we must map $x'_j \to x'_i x'_j$, where $x_i$ can be either $e_i$ or $m_i$ and $x_i x'_i = \varepsilon_i$. We also cannont map $e_i \to e_i m_i$ within a layer because this exchanges a boson with a fermion. In other words all symmetries of the model can be generated by the transformations

$$e_i \leftrightarrow m_i \tag{2.68}$$

$$e_i \leftrightarrow e_j \text{ and } m_i \leftrightarrow m_j \tag{2.69}$$

$$e_i \leftrightarrow e_i e_j \text{ and } m_j \leftrightarrow m_i m_j \tag{2.70}$$

which are simply the generators of all colour code symmetries generalised to act on a stack of more than two surface codes [Kesselring et al., 2018]. A simple way to obtain a twist corresponding to a $\beta_k$ anyon is simply to combine twists associated with symmetry (2.68) on $k$ different levels. The domain walls produced by these symmetries in the code correspond respectively to lines of H, SWAP and CNOT gates applied in the code stack. Since SWAP can be generated from CNOTs the generating set of symmetries can be reduced to just (2.68) and (2.70). This is consistent with the set of generating symmetries identified in [Webster and Bartlett, 2018b] although we arrive at this result by a different method. Any product of these symmetries thus corresponds to a product of Clifford gates in the code stack and so the code containing the twists will also be a 2D stabiliser code. Braiding operations are performed using predefined sets of single-qubit Pauli measurements and additional modifications to stabilisers by the Clifford gates listed above [Brown et al., 2017] and such operations in a 2D stabiliser code should not result in a logical non-Clifford gate. Thus all twists produced by composition of these symmetries should have Clifford braiding relations.

This result is valid for more than just self-inverse twists since (2.68)-(2.70) are the generators of *all* symmetries of the anyon model. Thus the restriction to Clifford braiding operations is valid for all twists in stacked surface codes. This is in agreement with recent results regarding the power of defect braiding in topological codes [Webster and Bartlett, 2018a][Webster and Bartlett, 2019].

Finally we comment briefly on the fault-tolerance of such braiding procedures. As mentioned above, braiding operations with twists can be performed using the standard code deformation techniques of (1) measurement of modified stabilisers and (2) single-qubit Pauli measurements to remove physical qubits from the code and provide information for decoding [Brown et al., 2017]. In a code with local stabilisers these operations will also be local so we expect that braids with these generalised twists should remain fault-tolerant under existing decoding procedures.

# Chapter 3

# Numerical Implementation of Just-In-Time Decoding in the 3D Surface Code

## 3.1 Introduction and Overview

In the previous chapter we discussed code deformations which implement logical operations directly. In this chapter we will instead focus on a way of using code deformation to enable, but not directly apply, logical operations which would otherwise not be possible. Additionally, we will be shifting our focus from a condensed matter style perspective on two-dimensional topological codes to an more error correction focused perspective on three-dimensional topological codes. In particular, we will be more concerned with decoding processes and fault tolerance than with quasiparticle excitations. We will discuss a dimension jumping process which, when combined with a just-in-time (JIT) decoder, allows for the implementation of a linear-time (in the code distance $d$) transversal $CCZ$ gate between three copies of the 2D surface code. These decoders were first proposed in [Bombín, 2018a] and adapted to the surface code in [Brown, 2020]. Our contribution is the construction of a new set of surface code lattices compatible with this procedure and the numerical demonstration of a threshold for JIT decoding in these lattices.

As discussed in chapter 1, a decoding algorithm is a classical process in a

quantum error correction scheme which takes as input a set of stabiliser measurement outcomes (referred to as a syndrome) and returns a correction operator. A correction is successful if the product of the correction operator and the original error is equivalent to a stabiliser. Of particular interest are *single-shot* decoding schemes where corrections can be inferred reliably from a single round of stabiliser measurements even in the presence of measurement errors [Bombín, 2015, Campbell, 2019, Quintavalle et al., 2021]. It is believed that single-shot decoding is not possible for 2D topological codes [Campbell, 2019] and so measurement errors must instead be counteracted by repeated rounds of stabiliser measurement, with fault-tolerance in a distance $d$ code requiring $O(d)$ repeats [Fowler et al., 2012]. As noted by Bombín in [Bombín, 2018a] this need for repeated measurements results in a discrepancy between constant-time circuits in 3D topological codes and linear-time circuits in 2D topological codes: in principle these both have spacetime cost $O(d^3)$, but in practise the 2D code will also require $O(d)$ measurement rounds between each set of operations, resulting in a time cost of $O(d^2)$ and a spacetime cost of $O(d^4)$.

Obviously lower resource costs for quantum computation are desirable, but the ability to use 2D rather than 3D topological codes is also desirable from an engineering perspective. It would therefore be ideal if the spacetime cost of $O(d^3)$ could be recovered in the 2D case. To this end, Bombín proposed the concept of a just-in-time (JIT) decoder [Bombín, 2018a] which supplies, at each timestep of the computational procedure, a best-guess correction based not only on the present syndrome of the code but also on the entire syndrome history. By interpreting the syndrome of a (2+1)-dimensional code as the syndrome of the corresponding (3+0)-dimensional code such a decoder allows for a form of pseudo-single-shot decoding of 2D codes, where measurements are not repeated and mistakes in the correction due to measurement errors are compensated for at later timesteps once the measurement errors that caused them are identified. The price for using such a decoder is that our 2D codes must be replaced with very thin slices of 3D code in which we can detect (but not reliably correct) measurement errors. The thickness of these slices is independent of $d$ and so while they are not strictly 2D (in the sense that they cannot

be embedded in a two-dimensional manifold) they still only require an architecture which is scalable in just two dimensions. In what follows, we will use the term "layer" to refer to a strictly 2D code and "slice" to refer to a bounded-thickness section of 3D code. We will also use bars over states and operators to refer to logical versions of these.

In its original formulation, Bombín used the idea of JIT decoding to circumvent causal restrictions encountered when attempting to translate a (3+0)-dimensional measurement-based computing scheme to a (2+1)-dimensional one. This scheme was based on the 3D colour code [Bombín and Martin-Delgado, 2006, 2007, Kubica and Beverland, 2015] and the ideas presented there were translated to the 3D surface code by Brown [Brown, 2020], who used JIT decoding to prove a threshold for a linear-time $\overline{CCZ}$ between three 2D surface codes. When combined with the Clifford group (which is also implementable in the 2D surface code [Fowler et al., 2012, Brown et al., 2017]) this provides us with a way to obtain a universal gate set which is potentially more efficient than competing techniques such as magic state distillation [Fowler et al., 2012, Bravyi and Kitaev, 2005, Litinski, 2019].

In this work we present a full implementation of Brown's procedure. In each of the three codes, we construct a scalable slice which is compatible with the various requirements of the procedure (discussed below). Having constructed these slices, we then simulate the performance of a simple JIT decoder in this setting and observe a threshold $p_c \sim 0.1\%$ in all three codes; see fig. 3.11. This is (to our knowledge) the first numerical demonstration of a threshold for JIT decoding.

In what remains of this section we provide an overview of our implementation of Brown's procedure and then discuss each component in more detail in subsequent sections.

The first such component is the 3D surface code (section 3.2). Unlike its 2D counterpart, this code admits a transversal three-qubit non-Clifford gate (the $CCZ$ gate) between three overlapping copies of the code [Kubica et al., 2015, Vasmer and Browne, 2019]. The aim of Brown's procedure is to use the equivalence between the 3D surface code in (3+0) dimensions and the 2D surface code in (2+1) dimensions

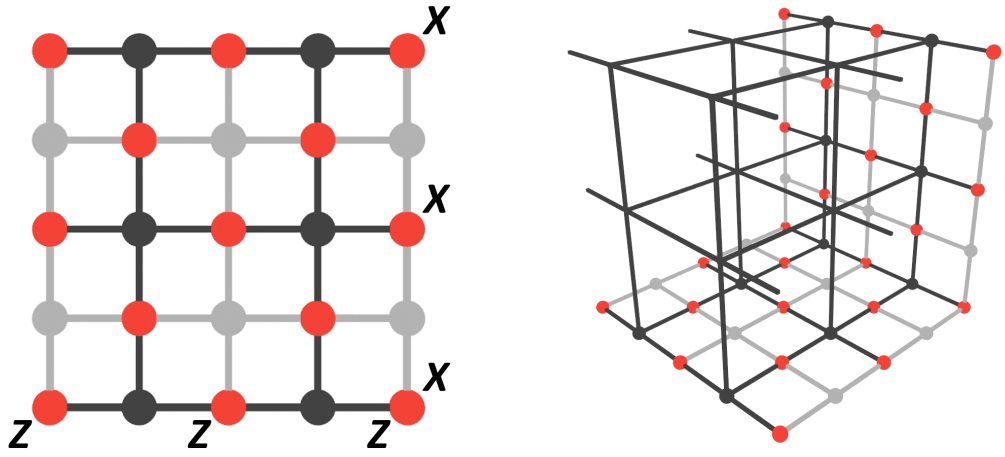
to implement a linear-time (in the code distance $d$) version of this gate between three overlapping 2D codes.

To achieve this we require a division of the three overlapping 3D codes into $O(d)$ bounded height slices which satisfy various requirements, the most important ones being that each slice must itself be a valid code with distance $d$, and that at each timestep the three codes should agree on a common set of qubits on which we should apply $CCZ$ (section 3.4). We also require a method of moving from one slice to the next, which cannot simply be "waiting" (despite the fact that we are using time as a dimension) because the overlap of the three 2D codes must be different at each timestep so they must move relative to each other. This is achieved by *dimension jumping* [Bombín, 2016] between the slice and the 2D layers on its top and bottom. $Z$ errors which arise during the procedure are also dealt with as part of this process (section 3.3). We must also ensure that all of these operations commute with the logical action of the $CCZ$ so that the entire procedure has the intended effect (section 3.5).

If the above can be done successfully then we will have obtained a procedure which, in the absence of errors, implements $\overline{CCZ}$ between three 2D surface codes in linear-time. To make the procedure fault-tolerant in the presence of $X$ and measurement errors we must use a just-in-time decoder (section 3.6). A single timestep of the full, fault-tolerant procedure then occurs as follows:

- Begin with three overlapping 2D codes/layers
- Expand to three overlapping slices of 3D code
- Apply JIT decoding operations to the three slices
- Apply $CCZ$ gates between the overlapping qubits
- Collapse back to three 2D layers

The layers we collapse to are on the opposite side of the slice from those we started in. It is possible to implement the entire procedure on an architecture which is only one slice thick by redefining our time direction at the start of each timestep.




(a) (b)

**Figure 3.1:** (a) A 2D surface code. $X$ stabilisers are on vertices of the dark lattice, $Z$ stabilisers are on faces and data qubits are on edges. Ancilla qubits used for stabiliser measurement are shown in dark and light for $X$ and $Z$ stabilisers respectively. Data qubits are shown in red. Weight three implementations of $\overline{X}$ and $\overline{Z}$ are also shown. (b) A 3D surface code. Qubits are placed on edges as in the 2D code, but to improve readability these qubits are only shown explicitly on the bottom and back-right boundaries.

## 3.2 The 3D Surface Code

We start by considering the 2D surface code [Kitaev, 2003, Bravyi and Kitaev, 1998] as in fig. 3.1. Here we are using the "Kitaev picture" where (with respect to the dark lattice) data qubits (red) are associated with edges. To each vertex of this lattice we associate a stabiliser generator $X(v) = \prod_{\{e|v\in e\}} X_e$ which acts with Pauli $X$ on all edges $e$ that meet at vertex $v$. To each face of this lattice we associate a stabiliser generator $Z(f) = \prod_{e\in f} Z_e$ which acts on all edges belonging to face $f$. Ancilla qubits used in measurement of these operators are also shown in fig. 3.1 (dark for $X$ and light for $Z$). Also shown in fig. 3.1 (in light grey) is the dual lattice, obtained by replacing the faces of the original lattice with vertices and the vertices with faces. In this lattice the Z stabilisers are on vertices and the X stabilisers are on faces.

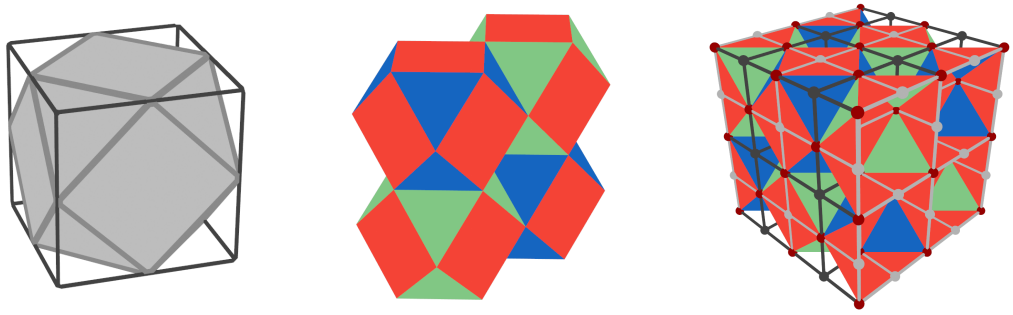
There are two types of boundary in this code which are commonly referred to as rough (left and right) and smooth (top and bottom). Logical $X$ operators are strings of single-qubit $X$ operators running between smooth boundaries while logical $Z$ operators are strings of single-qubit $Z$s running between rough boundaries. As such,

we will henceforth refer to smooth and rough boundaries as $X$ and $Z$ boundaries respectively.

Figure 3.1 shows how to obtain a 3D surface code [Dennis et al., 2002] from a 2D one, simply by extending the lattice into the third dimension. The dark square lattice has become a cubic one but we have retained the same assignment of parts of the code to geometric features of the lattice, i.e. data qubits are on edges, $X$ stabilisers are on vertices and $Z$ stabilisers are on faces. Four of the boundaries (top, bottom, front-left and back-right) of the code are $X$ boundaries and resemble 2D surface codes. $\overline{X}$ in this code is a sheet of single-qubit $X$ operators running between all four of these boundaries. The other two boundaries are $Z$ boundaries and $\overline{Z}$ is a string of single-qubit $Z$s running between these boundaries.

To see why the dimension of $\overline{X}$ has changed but the dimension of $\overline{Z}$ has not we can look at the structure of the stabiliser generators. In the 2D surface code each qubit is part of at most two $X$ and two $Z$ stabiliser generators so single-qubit errors of either type only violate a pair of generators. Longer strings of $X$ or $Z$ errors will commute with all generators along their length and only anticommute with a single generator at each end of the string. A logical operator corresponds to a string with both of its endpoints connected to the relevant boundaries where they cannot be detected. In 3D, as in 2D, every qubit is part of at most two $X$ generators and so logical $Z$ operators are once again strings. However, in the bulk of the cubic lattice four faces meet at every edge so qubits in the bulk are part of four $Z$ generators. This means that a string of $X$ errors will be detected not just by the generators at its endpoints but also by generators adjacent to the string along its entire length. In other words, the $Z$ stabiliser syndromes are loops around regions containing $X$ errors. This means that instead of $\overline{X}$ being a string with its endpoints connected to boundaries it must be a membrane with its entire perimeter connected to boundaries.

These loop-like $Z$ syndromes have another effect: they allow us to detect measurement errors. We know that valid syndromes should form closed loops, so syndromes not satisfying this property must have been produced (at least in part) by measurement errors. This allows us to repair these syndromes (by joining the



(a) (b) (c)

**Figure 3.2:** (a) Original cubic lattice (dark edges) and cuboctahedral cell of rectified lattice (light faces and edges). There will also be one octahedral cell around each vertex of the original lattice. (b) Four cuboctahedral cells of the rectified lattice. Half of an octahedral cell formed from the negative space of the cuboctahedra can be seen at the centre. Square faces are coloured red and triangular faces are coloured green or blue such that each cell only has two different colours of face. (c) Correspondence between distance-3 surface codes in the Kitaev (as shown in fig. 3.1) and rectified pictures. A correspondence can be seen between $Z$ stabilisers (faces of dark cubic lattice) in the Kitaev picture and red faces of the rectified lattice. There is also a correspondence between $X$ stabilisers (vertices) of the cubic lattice and octahedra in the rectified lattice (which contain no red faces).

endpoints of strings to form loops) and removes the need for repeated measurements of these stabilisers in a process termed *single-shot error correction* [Bombín, 2015, Campbell, 2019]. This stands in contrast to the 2D case where measurement errors can only be detected by repetition of these measurements.

We will use the rectified lattice picture of 3D surface codes to construct our slices since it allows us to describe all three codes using the same lattice [Vasmer and Browne, 2019]. This lattice is obtained from the standard cubic lattice by adding a new vertex at the middle of each edge, connecting these vertices if they belong to the same face and then deleting the original lattice. This results in one cuboctahedral cell per cubic cell of the original lattice (shown in fig. 3.2 (a)) and the negative space between these cells produces additional octahedral cells (half of such a cell can be seen at the centre of fig. 3.2 (b)). Figure 3.2 (b) also shows a colouring of the rectified lattice where each cell has two colours of face (red-blue and red-green cuboctahedra are visible and octahedral cells will be blue-green). It is then possible to simultaneously define three 3D surface codes on this lattice: three qubits are

associated with each vertex and one colour is associated with each code. $c$-faces (for $c \in \{r, g, b\}$) represent $Z$ stabilisers in the $c$ code and cells containing no $c$-faces represent $X$ stabilisers. An example is shown in fig. 3.2 (c).

## 3.3 Dimension Jumping in Surface Codes

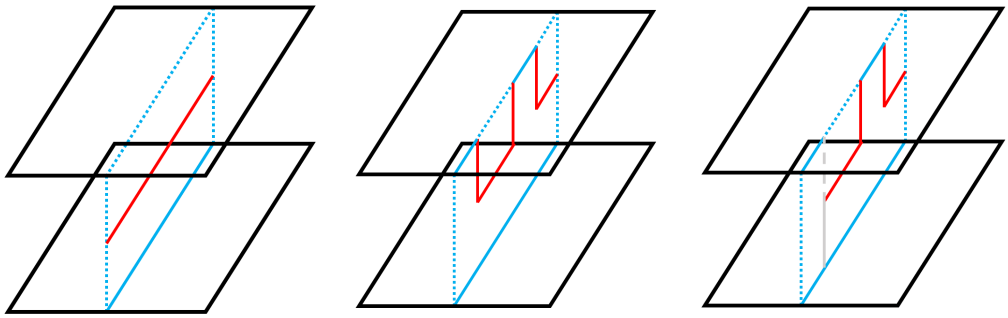
The ability to swap between a thin slice of 3D surface code and a 2D surface code (a process termed "dimension jumping [Bombín, 2016]") is at the heart of Brown's procedure. Similarly to JIT decoding, this process was originally studied by Bombín for use in the 3D tetrahedral colour code and was adapted for use in the surface code by Brown [Brown, 2020], although a process equivalent to the 3D $\rightarrow$ 2D collapse was previously studied by Raussendorf, Bravyi and Harrington in a measurement-based setting [Raussendorf et al., 2005]. Despite this, a comprehensive explanation of dimension jumping in surface codes did not exist in the literature (to our knowledge) prior to our work on this topic [Scruby et al., 2021].

### 3.3.1 2D to 3D expansion

As discussed in chapter 1, the expansion part of dimension jumping in the colour code is extremely straightforward:

- Start with a 2D colour code supported on a boundary of a 3D colour code.
- Measure stabiliser generators of the 3D colour code.
- Apply error correction to the 3D code.

The simplicity of this process is due to the fact that the 3D colour code minus this boundary encodes no qubits, so we do not need to worry about information from that code getting mixed with information from the 2D code. On the other hand, the 3D surface code minus a boundary still encodes a qubit and so we must take extra steps to ensure that the final logical state of the 3D code is the same as the initial logical state of the 2D code. Additionally, in the specific case where at each timestep we are combining dimension jumping with the application of $CCZ$ gates to part of the code we want our dimension jump to involve only measurements of $Z$



(a) (b) (c)

**Figure 3.3:** (a) Two 2D surface codes (black) which we imagine are entangled by measurement of intermediate stabilisers (not shown) to form a slice of 3D surface code. We imagine that we have applied a logical $\overline{X}$ to the lower code (solid blue line) but not to the upper code. This results in the red syndrome on the intermediate stabilisers. To successfully transfer the state of the lower 2D code into the 3D code we must apply a matching 2D logical $X$ to the top code which completes the logical $X$ of the 3D code (dashed blue). We can equivalently think of this as applying a correction which pushes the syndrome onto the top layer of the code. (b) In this example the the top code contains some strings of $X$ errors. The red syndrome now consists of loops joined to the top and side boundaries and "filling in" these loops to push them to the top boundary will once again correctly transfer the state of the lower 2D code to the 3D one. (c) Two measurement errors on a top code (dashed grey) and bottom code (solid grey) stabiliser cause us to lose track of what is inside and outside of a loop, making it difficult to reliably transfer the state of the 2D code into the bulk (the stabilisers are in the plane of the 2D code but the corresponding syndromes are perpendicular).

stabilisers. This is because the $X$ stabilisers do not all commute with this application of $CCZ$ gates and so the 2D codes at intermediate steps of the procedure will not be in eigenstates of these $X$ stabilisers. As such, measuring these stabilisers will change the state of the code and the overall effect of the procedure will not be a logical $CCZ$.

A suitable dimension jump is implemented by

- Start with a 3D surface code $\mathcal{C}$ and its boundary $\partial\mathcal{C}$, which is a 2D surface code in a state $|\overline{\psi}\rangle$. These codes must be chosen such that the $Z$ stabilisers of the 2D code commute with the $X$ stabilisers of both the 2D and 3D codes.

- Prepare all qubits belonging to $\mathcal{C} \setminus \partial\mathcal{C}$ in $|+\rangle$.

- Measure the $Z$ stabiliser generators of the 3D code that are not $Z$ stabilisers of the 2D code.

- Apply a correction which returns the 3D code to its codespace. This correction should not have support on any of the qubits of the original 2D code.

The preparation of new qubits in $|+\rangle$ combined with the commutivity requirement on the $Z$ stabilisers in the 2D code ensures that measurement of the 3D $Z$ stabilisers will project the encoded state either to a state in the codespace of the 3D code or to a state that can be returned to the codespace using a correction inferred from the measurement outcomes of these stabilisers. The distribution of $X$ errors that this correction addresses are what is referred to by Bombín as a Pauli frame [Bombín, 2018a] and by Brown as a gauge of the 3D code [Brown, 2020].

To understand why is it important not to measure the $Z$ stabilisers of the 2D code during the dimension jump and why the correction should have no support on this code it is easiest to consider an idealised example as in fig. 3.3. In (a) and (b) we see how applying corrections to the top layer stretches the string-like logical $\overline{X}$ of the lower 2D code into the sheetlike logical $\overline{X}$ of the 3D code and in (c) we see the issues that can arise if we allow for measurement of stabilisers of the initial 2D code during the jump (and thus allow for the possibility of measurement errors on these stabilisers). Note that in fig. 3.3 we begin with two surface codes and then entangle them via intermediate stabiliser measurements instead of initialising and entangling the top code in a single step by measuring top and intermediate stabilisers simultaneously. These two cases are equivalent since stabiliser measurements commute by definition. We do not need to do anything to ensure the correct transfer of $\overline{Z}$ from the 2D code into the 3D one because $\overline{Z}$ of either 2D code is a valid implementation of $\overline{Z}$ in the 3D code. By preparing the top code in $|\overline{+}\rangle$ (by preparing the physical qubits in $|+\rangle$ and measuring $Z$ stabilisers), we can ensure no logical $Z$ is applied in this code, so any implementation of $\overline{Z}$ applied to lower 2D code will be equivalently applied to the 3D code.

### 3.3.2 3D to 2D collapse

The collapse part of dimensional jumping in colour codes can similarly be adapted to surface codes. The steps of this process are

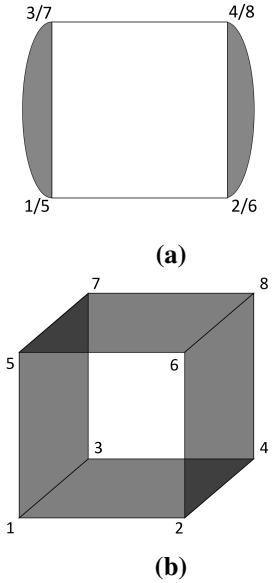



**Figure 3.4:** 2x2 and 2x2x2 minimal surface codes. Qubits are on vertices. (a) has two $X$ stabiliser generators (dark boundaries) and one $Z$ stabiliser (light face) so encodes one qubit. In (b) there is a $Z$ stabiliser on each face of the cube and an $X$ stabiliser on each of the dark boundaries. Only three of these $X$ and four of these $Z$ stabilisers are independent so this code also encodes a single qubit. A version of (a) exists on the top and bottom faces of (b). An implementation of $\overline{X}$ is supported on either of the light faces of the 3D code and on the top and bottom edges of the 2D code while $\overline{Z}$ is supported on either of the other two edges in the 2D code and on these same edges in the 3D code.

- Measure all qubits in $\mathcal{C} \setminus \partial\mathcal{C}$ in the $X$ basis.
- Apply a $Z$ correction to the qubits of $\partial\mathcal{C}$ based on the outcome of the $X$ measurements.

To understand how this correction is obtained it is easiest to consider a simple example, namely the one given in fig. 3.4. Here we have minimal examples of (a) a 2x2 and (b) a 2x2x2 surface code. We can switch between these two codes using dimension jumping since a version of (a) exists on the top and bottom faces of (b).

Consider the case where we begin in a logical state of the 3D code and measure qubits 1-4 in the $X$ basis. Since we were in a logical state the total parity of these four measurements must be even so either all four qubits were measured to be in the same state or two were in $|+\rangle$ and two were in $|-\rangle$. In the former case we will have projected to a logical state of the top 2D code but in the latter case will need to apply a correction. This is due to the fact that measuring the bottom face in $X$ projects us to a random configuration of the $Z$ stabilisers on the side faces and these stabilisers leave a "footprint" on the top code which may require a correction: for example, the restriction of $Z_1Z_2Z_5Z_6$ to the top face results in an error while the restriction of $Z_2Z_4Z_6Z_8$ is a logical $Z$ of the 2D code. Fortunately this correction is easy to find since the stabilisers leave an identical footprint on the bottom code, so we need only apply $Z$ to qubits in the top code wherever we measure $|-\rangle$ in the bottom code (in

more complicated geometries the correction is not quite so simple but can still be inferred straightforwardly from the stabiliser structure). This process is important not just for correcting errors but also for correctly transferring the logical state of the code, since in cases where bottom-face qubits measure $|-\rangle$ not due to a stabiliser footprint but due to an application of $\overline{Z}$, the correction applied to the top code will transfer this application of the operator from the bottom to the top. It is interesting to contrast this with the state-transfer procedure in the expansion step: there the transfer of $\overline{Z}$ from 2D to 3D was automatic and extra steps were required to properly transfer $\overline{X}$ but here the roles of $X$ and $Z$ are reversed (since any application of $\overline{X}$ to the 3D code is partially supported on a string corresponding to a representation of $\overline{X}$ in the 2D code).

In the case where there are $Z$ errors in the bottom code it is no longer possible to infer the necessary correction just from the measurement outcomes of the bottom face, but we can combine these measurements with measurements of the $X$ stabilisers of the 2D code to identify these errors (by reconstructing the syndrome of the 3D code). In our example, an odd parity of bottom face measurements tells us $X_1X_2X_3X_4$ would be violated, the parity of qubits 1 and 3 (2 and 4) together with the measurement outcome of $X_5X_7$ ($X_6X_8$) allows us to infer the measurement outcomes of the two side-face $X$ stabilisers and the product of $X_5X_7$ and $X_6X_8$ gives us the outcome of the top-face stabiliser. Because this code is only distance-2 we cannot correct for an error in this case, but in higher-distance codes we can.

An additional complication is introduced in the case where we apply *CCZ* gates to a subsection of the 3D code during each timestep. In this case we cannot measure the $X$ stabilisers of the 2D code post-collapse (the state we project to in the collapse will not be an eigenstate of these stabilisers), and so we cannot fault-tolerantly infer corrections for the top face. Fortunately, this is not an issue because $Z$ errors commute with *CCZ* and so applying a single $Z$ correction to the final code is equivalent to applying error correction throughout. This correction is obtained in two steps: firstly we must combine the $X$ stabiliser measurements from the final 2D code with single-qubit measurement outcomes from previous slices to obtain an $X$

stabiliser syndrome for a 3D surface code and decode this syndrome to identify the locations of $Z$ errors. Secondly, we combine these $Z$ error locations with the single-qubit measurement outcomes to obtain a corrected set of single-qubit measurement outcomes, and from these we calculate a correction for the final 2D code. This operator can be interpreted as the adaptive $Z$ correction of a teleportation circuit, and this perspective on the procedure is discussed further in [Webster et al., 2021].

## 3.4 Constructing Slices

In this section we present our proposed slices through the three 3D surface codes. We begin with an examination of the criteria which these slices must satisfy, then discuss each of our slices individually and finally demonstrate that they have the correct overlap at each step of the procedure.

### 3.4.1 Criteria for Valid Slices

We now examine some necessary criteria which slices through 3D codes must meet in order to be used in Brown's procedure. This list is by no means exhaustive, but serves to highlight some of the major issues which must be avoided when constructing slices.

Firstly, we have the requirement that any representative of a logical operator in the slice must have weight at least $d$ in order to preserve the distance of the code. The ability to construct slices satisfying this relies on a specific property of the 3D surface code (the string-like logical operator of code is required to run between a particular pair of boundaries) which is not present in general 3D codes. This restriction on the string-like logical means that as long as we ensure that its associated boundaries are on the sides of the slice and not on the top and bottom we are guaranteed logical operators with weight at least $d$. It is this restriction that forces at least one of the three codes to have a different time direction to the other two, since the string-like logical operators of three 3D surface codes which admit a transversal $CCZ$ are all perpendicular [Vasmer and Browne, 2019] so there is no way to define a consistent time direction for all three (in three dimensions) such that slices in all three codes have this property.

Secondly, we require that all $Z$ stabilisers in the layers commute with all $X$ stabilisers in the slices. This requirement is due to a combination of the dimension jumping process and the $CCZ$ gate as discussed in section 3.3. In addition, we require that the support of a 3D $X$ stabiliser on a layer corresponds exactly to a 2D $X$ stabiliser within that layer. This is what ensures that measurements of $X$ stabilisers in the final layer alongside the single-qubit measurement history throughout the procedure can reproduce the full $X$ stabiliser syndrome of a 3D surface code. This rule prevents us from having things such as a 3D $X$ stabiliser which is supported on only a single qubit of a layer.

Finally, all three slices must agree on a common set of overlapping qubits at every timestep. Stated differently, this means that qubits between which we intend to apply $CCZ$ must all be live simultaneously. This is essential for the procedure or the three codes would disagree on which qubits $CCZ$ should be applied to at each timestep.

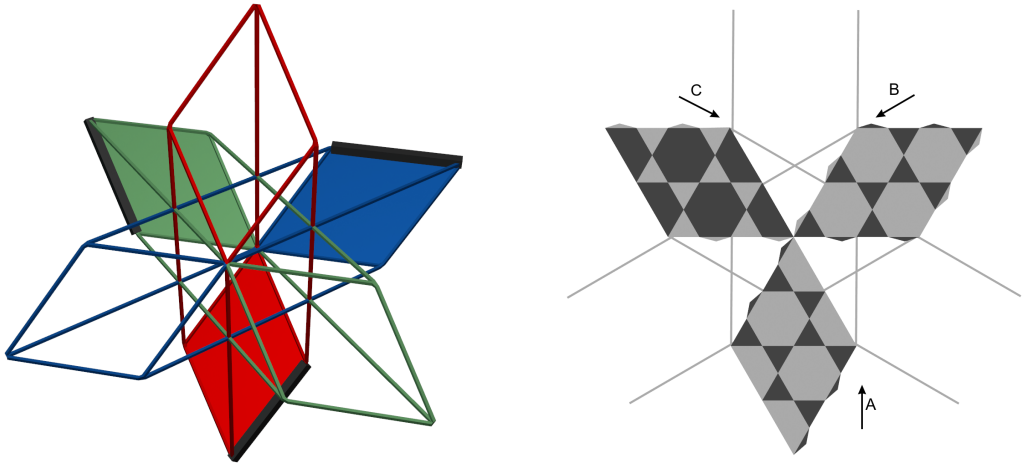
The structures originally proposed by Brown were "staircases" through the cubic lattice in the Kitaev picture. In this section we present an alternate set of lattice slices which we find easier to understand and simulate, although we believe equivalent simulations could be performed using the staircase slices.

### 3.4.2 Proposed Layers and Slices

The layers we propose are shown at a macroscopic and microscopic level (for distance 3) in fig. 3.5 (a) and (b) respectively. In Brown's original proposal the time direction was the same for two of the codes and different for the third, but here we use a different time direction for each code. This is still implementable in 2D (also shown in fig. 3.5 (b)). In the following subsections we show a distance-3 example of the corresponding slice through each of the three 3D codes and then discuss their overlap.

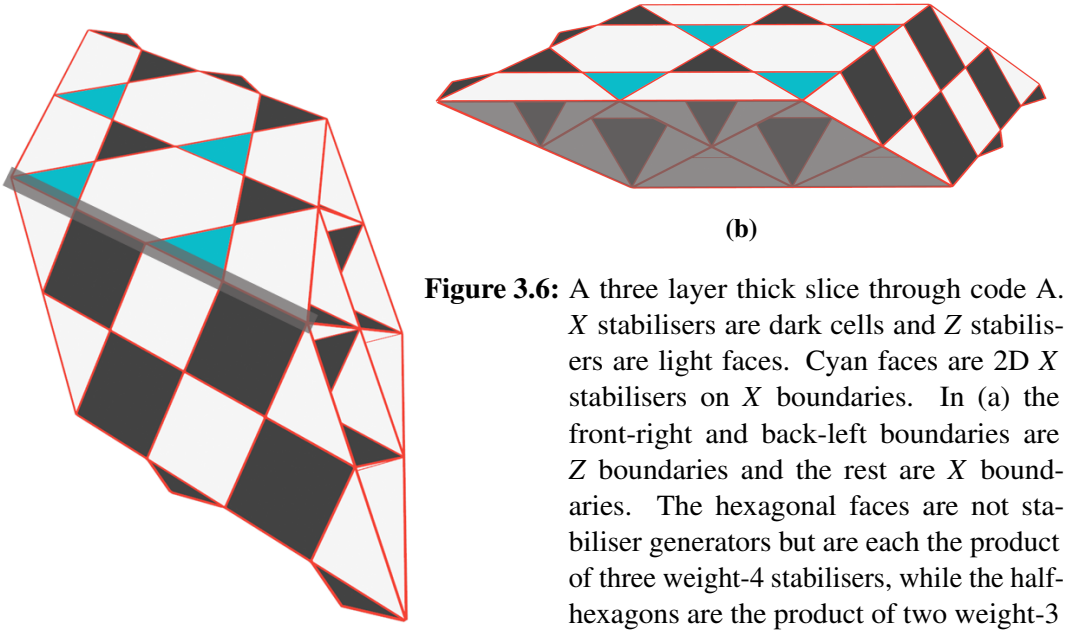
#### 3.4.2.1 Code A

This is the simplest of the three codes and corresponds to the red code on the rectified lattice. As illustrated in fig. 3.2 (c) this is the standard 3D surface code defined



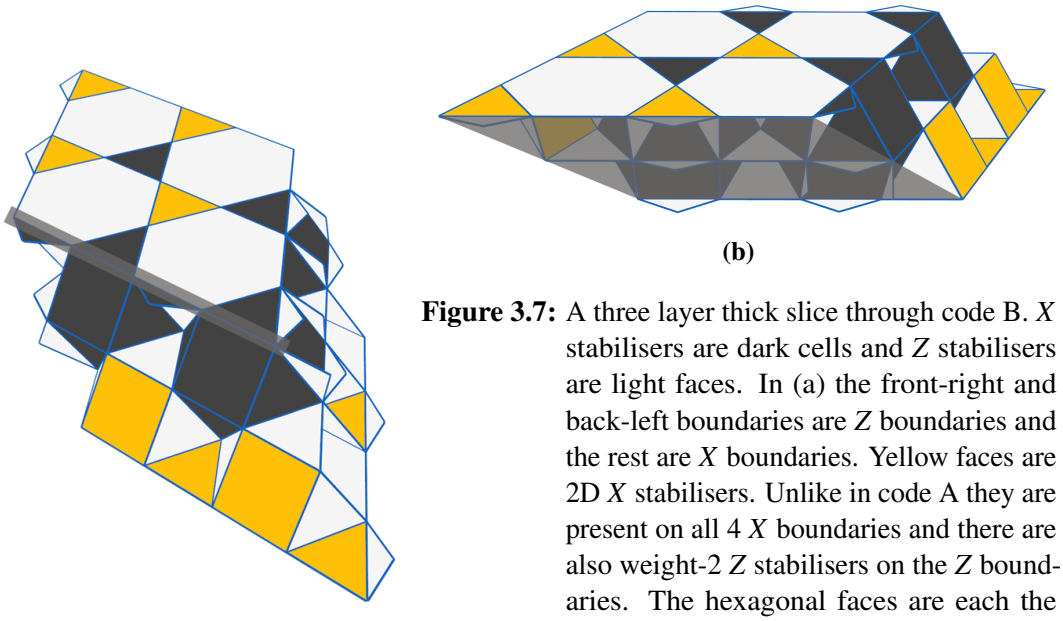

(a) (b)

**Figure 3.5:** (a) 3D spacetime for all three surface codes. Each of the three 2D codes sweeps out a 3D surface code over time and the cubic region where all three of these codes intersect supports a transversal $\overline{CCZ}$. (b) Microscopic details of three distance-3 2D surface codes (light faces are $Z$ stabilisers and dark faces are $X$ stabilisers, qubits are on vertices) and and their directions of motion during the procedure. Logical $Z$ operators for the 2D layers are shown in black for each code in (a) and by comparing with (b) we can see that they run between $Z$ boundaries.



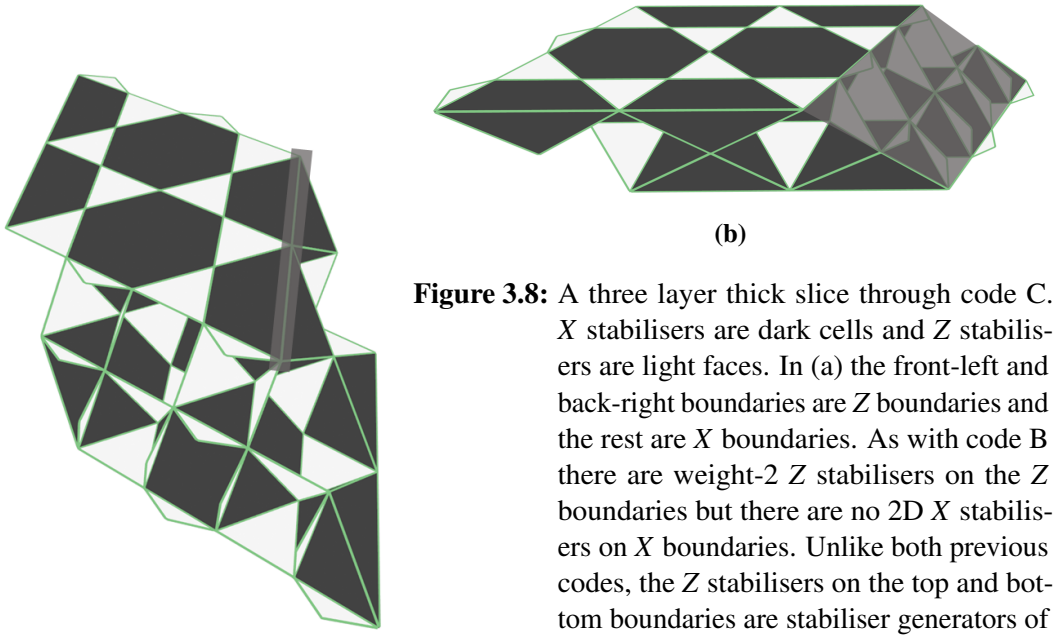
(a) (b)

**Figure 3.6:** A three layer thick slice through code A. $X$ stabilisers are dark cells and $Z$ stabilisers are light faces. Cyan faces are 2D $X$ stabilisers on $X$ boundaries. In (a) the front-right and back-left boundaries are $Z$ boundaries and the rest are $X$ boundaries. The hexagonal faces are not stabiliser generators but are each the product of three weight-4 stabilisers, while the half-hexagons are the product of two weight-3 stabilisers. A representative of logical Z/X is shown in grey in (a)/(b).



(a) (b)

**Figure 3.7:** A three layer thick slice through code B. $X$ stabilisers are dark cells and $Z$ stabilisers are light faces. In (a) the front-right and back-left boundaries are $Z$ boundaries and the rest are $X$ boundaries. Yellow faces are 2D $X$ stabilisers. Unlike in code A they are present on all 4 $X$ boundaries and there are also weight-2 $Z$ stabilisers on the $Z$ boundaries. The hexagonal faces are each the product of four weight-3 stabiliser generators. A representative of logical Z/X is shown in grey in (a)/(b).



(a) (b)

**Figure 3.8:** A three layer thick slice through code C. $X$ stabilisers are dark cells and $Z$ stabilisers are light faces. In (a) the front-left and back-right boundaries are $Z$ boundaries and the rest are $X$ boundaries. As with code B there are weight-2 $Z$ stabilisers on the $Z$ boundaries but there are no 2D $X$ stabilisers on $X$ boundaries. Unlike both previous codes, the $Z$ stabilisers on the top and bottom boundaries are stabiliser generators of the 3D code but those of the middle layer are not. A representative of logical Z/X is shown in grey in (a)/(b).

on a simple cubic lattice. Examples of the boundaries are shown in fig. 3.6 for a distance-3 slice. This slice is three layers thick, but it is actually possible to define a two-layer-thick slice in this code since the middle layer in fig. 3.6 is identical to the top and bottom layers. This is not true for the other two codes, and so we must also use a three-layer-thick slice here to ensure the correct overlap of the three slices.

#### 3.4.2.2 Code B

Code B is the blue code on the rectified lattice and a slice through this code is shown in fig. 3.7. The $Z$ stabilisers of this code are weight-3 (in contrast to the weight-4 $Z$ stabilisers of code A) while its $X$ stabilisers are weight-12 (in code A they were weight 6). Because the lattice constant for the blue code is twice that of the red code we must use a slice that is three layers thick.

It is interesting to note that the top 2D code for this slice differs from that of code A only by a half-hexagon translation on the kagome lattice and so one might imagine that it would be possible to define a slice in code A with top and bottom 2D codes matching those seen in code B. However, the weight-2 $Z$ boundary stabilisers present in these 2D codes do not commute with the 3D $X$ stabilisers of code A and so it is not possible to expand from the 2D code to the 3D one by only measuring $Z$ stabilisers.

#### 3.4.2.3 Code C

Code C is the green code on the rectified lattice and a slice through this code is shown in fig. 3.8. The top and bottom layers in this slice resemble the middle layer from the slice for code B, with $Z$ stabilisers on triangles and $X$ stabilisers on hexagons. The same is true for the middle layer of this slice and the top and bottom layers of the slice through code B. Additionally, these 2D codes are translated by a half-hexagon on the kagome lattice relative to those of code B and so have boundaries which match those of the 2D codes from code A. The inversion of stabilisers in these 2D codes relative to codes A and B mean that the weights of the logical operators are also exchanged (2D codes A and B had weight 3 $\overline{X}$ and weight 5 $\overline{Z}$ whereas here it is the opposite). Due to this, we observe better performance during JIT decoding in this code than in codes A and B (fig. 3.11), since the JIT decoder only deals with $X$

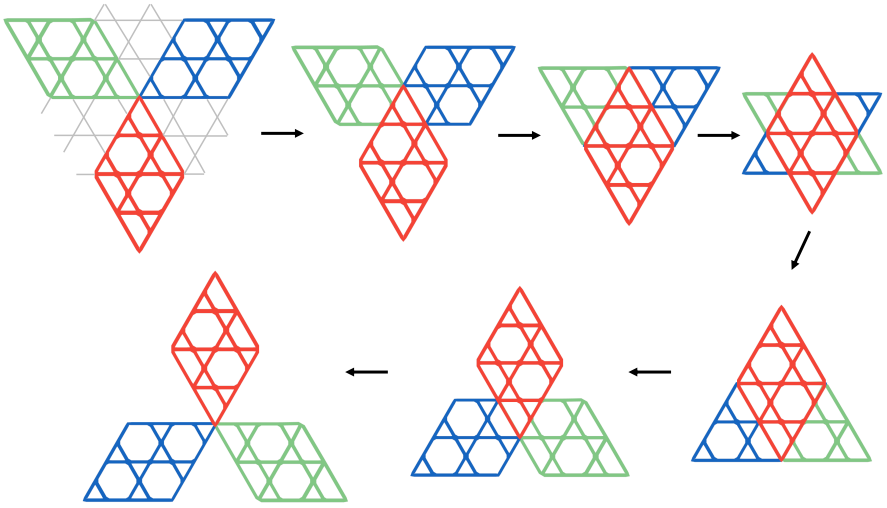


**Figure 3.9:** The three distance-3 2D surface codes from fig. 3.5 (b) passing through each other. The vertices (qubits) of a given code always coincide with the vertices of the other two codes in the overlapping region. A single, global kagome lattice can be consistently overlaid on all three codes at all points of the procedure, and the three individual codes correspond to sections cuts from this lattice, parts of which are shown in grey in the first image.

errors and Z errors are dealt with separately.

### 3.4.3 Overlap of the Three Codes

The full sequence of the distance-3 2D codes passing through each other can be seen in fig. 3.9. Vertices of all three codes coincide in the overlapping region, and in fact, these three codes can all be thought of as sections cut from the same lattice. If our slices were two layers thick then our three 2D codes would follow this sequence exactly, but because they are three layers thick we actually move two steps in this sequence for every step of the procedure, i.e. if we start with the first code configuration then after one cycle of expand $\rightarrow$ correct $\rightarrow$ *CCZ* $\rightarrow$ collapse we will have the third configuration. While we do not observe the second configuration in fig. 3.9 in a purely 2D setting it will correspond to the overlap of the three middle layers of the slices during the aforementioned set of operations. All qubits in each slice belong to one of the three layers and so all three layers agree on common sets of overlapping qubits.

### 3.4.4 Practical Implementation

This procedure would not be of much use if it were necessary to sweep through a physical 3D code. In Brown's proposal the time direction is changed at each timestep so that the physical architecture only needs to be one slice thick and we can move up and down between the top and bottom layers. Our slices are compatible with this, although the architecture must accommodate three layers instead of two. Additionally there must be room for two of the codes to move relative to the third (in the original proposal only one code needed to move) and this will increase the qubit cost of the procedure.

## 3.5 Linear-Time CCZ

Three 3D surface codes defined on a $d \times d \times d$ cube in the rectified lattice with boundaries as in fig. 3.2 admit a transversal $\overline{CCZ}$ gate [Vasmer and Browne, 2019]. Key to Brown's procedure is the idea that this gate is also implementable (in time linear in the code distance $d$) on three 2D surface codes whose overlap in spacetime is equivalent to such a 3D surface code. More precisely, as the three codes move through each other (via the dimension jumping process described in previous sections) *CCZ* gates are applied between all overlapping triples of physical qubits that do not belong to the top layer. The exclusion of top-layer qubits is necessary because these will become the bottom-layer qubits of the next slice and we only want to apply *CCZ* once to each qubit in the overlapping region. The intuition for why this process should work is that JIT decoding allows us to fault-tolerantly exchange a spatial dimension for a temporal one, so any process which gives a logical operation in $(3+0)$ dimensions should result in an equivalent logical operation in $(2+1)$ dimensions. However, some readers may not be satisfied with this argument and so it is both interesting and worthwhile to examine the workings of this gate in more detail.

To begin with, we neglect JIT decoding entirely and just consider the full 3D spacetime as shown in fig. 3.5a. The first thing to note about the spacetime of the three codes is that it is not a $d \times d \times d$ cube; rather, it is three parallelepipeds

whose intersection is a $d \times d \times d$ cube. It is not immediately obvious, and nor is it generally true, that applying *CCZ* transversally in this overlapping region should implement a logical $\overline{CCZ}$ between the codes. Recall that $\overline{CCZ}$ maps $\overline{X}_i$ to $\overline{X}_i\overline{CZ}_{jk}$ (where $CZ = 1/2(II + IZ + ZI - ZZ)$) and thus the intersection of $\overline{X}$ in any code with the region where we apply *CCZ*s must be a region supporting valid implementations of $\overline{Z}$ in the other two codes. In a version of fig. 3.5a where $\overline{Z}$ in each of the three codes runs in the "long" direction, the transversal application of *CCZ* between the overlapping qubits in the central cubic region will not implement $\overline{CCZ}$ because there is no implementation of $\overline{Z}$ in any of the three codes which is fully supported inside the cube. Fortunately this is not the case for the arrangement of codes described in the previous section, where instead $\overline{Z}$ runs in one of the "short" directions in each code (as shown by the dark lines in fig. 3.5a) and so each code possesses valid implementations of this operator which are contained within the cube. In contrast, there is no implementation of $\overline{X}$ contained within the cube in any of the codes, but this is not a problem since nothing is mapped to $\overline{X}$ by $\overline{CCZ}$. We require only that the intersection of $\overline{X}_i$ with the cube supports valid implementations of $\overline{Z}_j$ and $\overline{Z}_k$ and this requirement is satisfied. This means that a constant-time version of the procedure where we start with our initial set of 2D codes, use dimension jumping to expand to the full 3D spacetime, apply *CCZ*s to the overlapping region then collapse to the final set of 2D codes will implement the desired logical operation.

Now that we are satisfied that a single-timestep version of the procedure implements $\overline{CCZ}$ we can convince ourselves of the same thing for the linear-time version using induction. Consider the following two sets of operations, where "expand" should be understood to mean "measure 3D Z stabilisers then apply JIT decoding and *X* error correction" and we assume that these correction operations are successful:

- **A:** Begin with a layer at position $x$, expand to a slice with thickness $\Delta x$, apply *CCZ*s, collapse to a layer at position $x + \Delta x$, then repeat to end at a layer at position $x + 2\Delta x$.

- **B:** Begin with a layer at position $x$, expand to a slice with thickness $2\Delta x$, apply

*CCZ*s and collapse to a layer at position $x + 2\Delta x$.

Each qubit experiences the same set of operations in both cases: measurement of associated $Z$ stabilisers and an $X$ correction if one is required (for qubits not in the initial layer), application of *CCZ* (for qubits in the overlapping region and not in the final layer) and measurement in the $X$ basis (for qubits not in the final layer). The only way for this sequence of operations to be inequivalent in **A** and **B** is if operations in the first slice in **A** influence our choice of $X$ correction for qubits in the second slice in a way that does not occur in **B**. Incorrect choices of $X$ error correction operator in the first slice of **A** will have this effect but we are assuming that all correction attempts are successful[1]. The only other operation applied to the qubits of the first slice of **A** is *CCZ*, but this is not applied to qubits that are part of the second slice and its only effect will be to change the outcomes of our single-qubit measurements. As discussed in section 3.3, the correction inferred from these measurements is only applied at the very end of the procedure so we conclude that none of the operations on the first slice of **A** can influence the operations on the second and the logical effects of **A** and **B** must be equivalent. This conclusion, combined with the knowledge that performing the entire procedure in a single timestep correctly implements $\overline{CCZ}$, allows us to infer that the linear-time version of the gate also implements $\overline{CCZ}$.

## 3.6 The Delayed Matching Decoder

### 3.6.1 Description

Now that we have constructed a set of slices we are ready to consider explicit error correction operations within these slices. We imagine that we are at a point of the procedure where we have prepared the new qubits and measured the new stabilisers but have not yet applied any corrections. We have not remeasured the bottom-layer stabilisers as we assume that our choice of correction in the previous timestep was successful and all of these stabilisers are in the +1 eigenstate. The $Z$ stabiliser measurements result in a random distribution of $X$ errors on the new qubits and in a

[1] This will not be true in reality so this argument does not guarantee a threshold for the procedure. However, we are currently only asking if the gate has the correct logical action in principle and so these assumptions are justified.

syndrome for these errors consisting of a set of loops in the bulk or connected to the top or side $X$ boundaries. If there were errors on the bottom layer of qubits or if we had errors in some of our stabiliser measurements we will also have broken strings with endpoints which must be matched up to produce a valid syndrome. The delayed matching decoder is a modified version of the minimum-weight perfect matching (MWPM) decoder [Dennis et al., 2002, Edmonds, 1965, Fowler, 2014] and provides a simple but fault-tolerant method of performing this matching. This decoder was first proposed by Brown in [Brown, 2020] and its operation is is as follows:

**Setting:** In the slices presented in the previous section the qubits were on vertices and $Z$ stabilisers were on faces meaning the syndrome from these stabilisers will be a set of (possibly broken) loops on edges dual to these faces[2]. Endpoints of broken loops will be at the centres of cells and the distance between a pair of endpoints is measured in terms of path length on the dual edges connecting them[3] When we speak of joining pairs of endpoints to each other or to side boundaries we mean that the stabiliser measurement outcomes are flipped (in software) along a minimal path connecting this pair of objects.

**Inputs:**

- A vector $e$ of endpoint locations within the current slice
- A map (i.e. a set of key-value pairs) $M$ from pairs of endpoint locations (or an endpoint location and a boundary) to a "pseudodistance" between the elements of the pair (this pseudodistance will initially be equal to the true distance between the endpoints but will get smaller each time the endpoint pair recurs).
- An integer $c$ which specifies the pseudodistance below which it is permissible to join pairs of endpoints.
- An integer $r$ which specifies the amount by which we should reduce the pseudodistance between a pair of endpoints if we do not choose to join them.

[2]We recall that the dual of a 3D lattice is obtained by placing new vertices at the centres of cells of the original lattice, connecting these vertices if their cells share a face in the original lattice and then deleting the original lattice. This transformation maps cells to vertices, faces to edges and vice versa.

[3]Some readers may prefer to think in terms of the qubits-on-faces picture where $Z$ stabilisers are on edges and endpoints are on vertices. For code A the relevant lattice is the simple cubic lattice, whereas for codes B and C it is the rhombic dodecahedral lattice.

**Subroutines:**

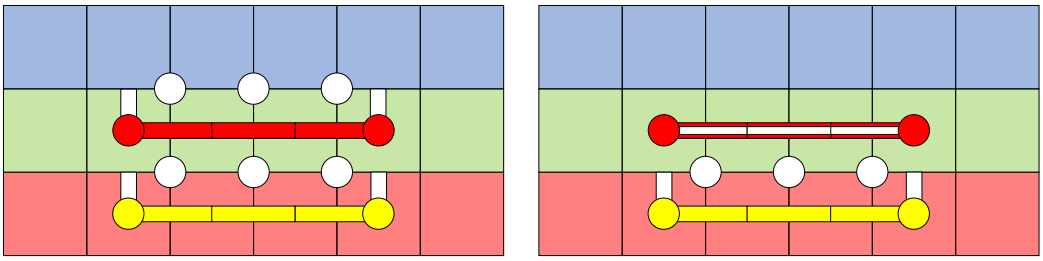
- A standard MWPM decoder MWPM which is allowed to match endpoints to each other and to the side $X$ boundaries but not the top or bottom $X$ boundaries. It takes a vector of endpoint locations as an argument and returns a vector of endpoint location pairs (or endpoint locations and boundaries).
- A function $d$ which takes as argument an endpoint location pair (or an endpoint location and a boundary) and returns the (true) distance between them.
- A function $t$ which takes as argument an endpoint location pair (or an endpoint location and a boundary) and returns the corresponding pair of locations in the slice for the next timestep.

**Algorithm 1** DELAYED MATCHING DECODER

**Input:** $e, M$
**Output:** None

$currentPairs \leftarrow \mathsf{MWPM}(e)$
**for** $pair \in M$ **do** ▷ $M$ update step 1
    **if** $pair \notin currentPairs$ **then**
        Remove $pair$ from $M$
    **end if**
**end for**
**for** $pair \in currentPairs$ **do** ▷ $M$ update step 2
    **if** $pair \notin M$ **then**
        $M[pair] \leftarrow d(pair)$
    **end if**
**end for**
**for** $pair \in M$ **do** ▷ Syndrome repair step
    **if** $M[pair] \leq c$ **then**
        Join the elements of $pair$ to each other
    **else**
        Join all endpoints in $pair$ to the top boundary
        $M[t(pair)] \leftarrow M[pair] - r$
    **end if**
    Remove $pair$ from $M$
**end for**

Once this algorithm is complete we should have a valid $Z$ stabiliser syndrome consisting entirely of unbroken loops. We then need to use this syndrome to infer a correction on the data qubits which will push these loops to the top boundary as in



(a) (b)

**Figure 3.10:** 1+1 dimensional cross-sections through (idealised) slices of a 3D surface code. Qubits are on vertices, $Z$ stabilisers are on edges (with syndromes on dual edges) and endpoints are on faces. The past, present and future are shown in red, green and blue respectively. (a) Issues caused by attempting to use a standard decoder in a sliced surface code. Three measurement errors (yellow lines) in the past caused two endpoints (yellow circles) and the decoder matched these to the top layer (white lines). An $X$ "correction" was then erroneously applied to three data qubits (white circles) due to this incorrect matching of endpoints. In the present, these $X$ errors cause the red syndrome and because bottom-layer stabilisers are not remeasured we get the two red endpoints. These will be matched to the top layer as before and the cycle will repeat, causing an error of unbounded size extending into the future. (b) The action of the delayed matching decoder on the same error. Once again endpoints in the past are matched to the top face, but in the present the decoder identifies that the same two endpoints occurred in the previous step and chooses to match them to each other instead of the the top.

fig. 3.3. We find that the sweep decoder [Kubica and Preskill, 2019, Vasmer et al., 2021] is a natural fit for this problem as its action is to generate corrections which push syndrome loops in a given direction, but in principle any valid decoder for loop-like syndromes could be used (e.g. [Breuckmann et al., 2016, Breuckmann and Ni, 2018, Duivenvoorden et al., 2019, Aloshious and Sarvepalli, 2021, Panteleev and Kalachev, 2019, Roffe et al., 2020, Quintavalle et al., 2021]).

It may be helpful at this point to consider the simple example presented in fig. 3.10. In this case we imagine that $c = r = 2$. At the start of the procedure (in the red slice) $e$ contains the two yellow endpoints and $M$ is empty. MWPM will pair the two endpoints to each other since it is not permitted to match them to the top or bottom boundaries and the side boundaries are too far away to be favourable. In the first update step nothing happens and in the second update step the pair of endpoints are added to $M$ with an associated value of 3 (the distance between them on dual

edges). In the syndrome repair step we compare $c$ to the cost of joining the pair to each other ($M[pair] = 3$) and since the former is smaller we join the pair to the top. A new pair of endpoints $t(pair)$ which corresponds to the yellow pair translated into the next slice (i.e. the two red endpoints) is added to $M$ with an associated value of $M[pair] - r = 3 - 2 = 1$ and $pair$ is removed from $M$.

At the start of the next step (in the green slice) $e$ contains the two red endpoints and $M$ contains this pair of endpoints with an associated pseudodistance of 1. MWPM will once again match this pair to each other and nothing happens in either update step because the pair of endpoints in $M$ exactly matches the output of MWPM. In the syndrome repair step we now have $M[pair] = 1$ which is less than $c$ so we join the two red endpoints to each other.

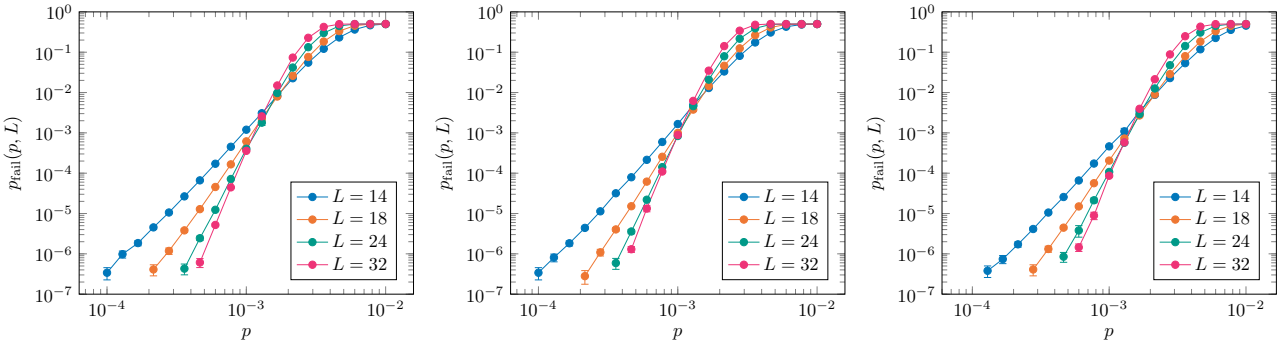
### 3.6.2 Numerical Implementation

We numerically investigate the performance of the delayed matching decoder for all three surface codes, using the slices described in section 3.4. The MWPM part of the simulation uses Kolmogorov's Blossom V implementation [Kolmogorov, 2009]. The numerical results presented here correspond to the case where we perform the expand $\rightarrow$ decode $\rightarrow$ collapse parts of the procedure but do not apply *CCZ*s between the codes. As such, we include both measurement errors on $Z$ stabilisers and physical $X$ errors in our simulation but not $Z$ errors or the $CZ$ errors, which arise due to application of $CCZ$ to qubits with $X$ errors.

For each of the three codes simulate the procedure for different values of $X$ and measurement error rate, $p$, and code distance, $L$. We count the number of logical errors $f$ that occur in $n$ trials and estimate the logical error rate $p_{\text{fail}}(p,L)$. The uncertainty in this value is calculated using the Agresti-Coull confidence interval [Agresti and Coull, 1998] which is more stable at low error rates than the standard confidence interval based on the normal distribution. Our estimate of $p_{\text{fail}}(p,L)$ is then:

$$p_{\text{fail}}(p,L) = \tilde{p} \pm 2\sqrt{\frac{\tilde{p}(1-\tilde{p})}{\tilde{n}}}, \tag{3.1}$$

where $\tilde{n} = n + 4$ and $\tilde{p} = \tilde{f}/\tilde{n}$ (with $\tilde{f} = f + 2$). This is approximately the 95%




**(a)** Code A **(b)** Code B **(c)** Code C

**Figure 3.11:** Numerical evidence of an error threshold for the delayed matching decoder. We ran Monte Carlo simulations for the three codes described in section 3.4. For each code, we estimate the logical error rate $p_{\text{fail}}(p,L)$ as a function of the $X$ error and measurement error rate, $p$, for different code distances $L$. The error threshold, $p_c$, is the value of $p$ where the curves for different $L$ intersect. We observe an error threshold of $p_c \sim 0.1\%$ in each of the three codes.

confidence interval.

In fig. 3.11 we plot $p_{\text{fail}}$ against $p$ for different values of $L$ for each code. The error threshold, $p_c$, is the point where the curves for different $L$ intersect and we observe evidence of an error threshold in the region of $p_c \sim 0.1\%$ for all three codes. This threshold estimate is many orders of magnitude larger than the theoretical value of $\sim 10^{-17}$ obtained by Brown in his proof [Brown, 2020]. In addition, our value is only one order of magnitude smaller than the 2D surface code threshold [Raussendorf and Harrington, 2007, Wang et al., 2011, Fowler et al., 2012, Stephens, 2014]. We anticipate that the performance of the delayed matching decoder could be significantly improved by using more of the syndrome history, so it is still possible that the optimised JIT decoding threshold could be competitive with the 2D surface code threshold. We observe improved suppression of $p_{\text{fail}}$ below threshold in code C when compared to codes A and B. This is consistent with the fact that, for a given value of $L$, $\overline{X}$ is higher weight in the layers for this code than in the other two.

## 3.7 Discussion

Now that we have covered each component of Brown's procedure in detail, it is helpful to once again provide an overview of the procedure so that we can see how

each component fits together. A single timestep proceeds as follows:

1. Begin with three 2D surface codes. All $Z$ stabilisers of these codes are assumed to be in the +1 eigenstate.

2. Expand to three thin slices of 3D surface code by preparing new data qubits in $|+\rangle$ and then measuring the new $Z$ stabilisers. The previously existing 2D codes will now be the bottom layers of these slices, and their stabilisers are *not* remeasured.

3. If our assumption regarding the states of the initial $Z$ stabilisers was correct and there are no measurement errors on the newly measured ones then the syndrome from these stabilisers will consist entirely of loops in the bulk or connected to the top or side boundaries. A correction that pushes all these loops to the top boundary will transfer the original state of the 2D code into the slice. If there are measurement errors on the newly measured stabilisers or X errors on the lower layer we will have an invalid syndrome containing broken strings, the endpoints of which must be paired up to produce a valid syndrome from which we can infer a correction. Incorrect pairings will result in X errors in the slice. This is the JIT decoding step.

4. Apply CCZs between all triples of qubits in the region where the three slices overlap except for those in the top layer. If there are X errors in this region due to incorrect decoding in the previous step then these will cause CZ errors on the other two codes.

5. Measure out all non-top layer qubits in the X basis. The output of these measurements will be fed to a global Z error decoder once the procedure is complete and used to find a Z error correction for the final code.

We have verified the integrity of the majority of the components involved in this process. The only part which is missing is the simulation of the Clifford errors arising from the non-Clifford gate and the numerical demonstration of a threshold for $Z$ errors. These errors will be the topic of the next chapter, but we wish to emphasize

that while they may affect the overall threshold for the procedure, they will not affect the JIT decoding thresholds shown here. This is because the JIT decoder only deals with $X$ errors and the errors arising from the $CCZ$ gate are $CZ$ errors which will be projected to a distribution of $Z$ errors by the single-qubit measurements which collapse the slice.

In contrast, the locations of the $CZ$ errors depend on the locations of $X$ errors post-JIT decoding, so while the distribution of $Z$ and $CZ$ errors will not affect the performance of the JIT decoder the reverse is not true. For this reason the development of more sophisticated and effective JIT decoders is also an important direction for future research. The delayed matching decoder has the advantage of being relatively simple, but it uses only a small amount of the information available in the syndrome history and it is reasonable to expect that significant improvements could be made to decoder performance by utilising more of the available information.

Another natural direction for future research would be the construction of similar slices in other topological codes; for example, one might want to perform an analogous procedure in the 2D/3D colour code in order to perform a linear-time logical T gate. However, this is not quite as straightforward as one might hope because, unlike in the 3D surface code, the string-like logical operator of the 3D tetrahedral colour code is not required to run in any particular direction and any edge of the tetrahedron supports a valid implementation of this operator. This means that any slice of bounded height through the 3D code will contain edges of bounded length which support low-weight logical operator implementations. Cubic colour code constructions encoding multiple logical qubits as in [Kubica et al., 2015] can avoid this problem for some but not all logical qubits. In these codes, as with three surface codes admitting a transversal $CCZ$, the string-like logical operators are all perpendicular, but unlike in the surface code case we cannot assign different time directions to different logical qubits. This means that for any choice of slice the string-like logical for one of the encoded qubits will run between the top and bottom boundaries. Additionally, this logical qubit will be lost completely in the collapse from 3D to 2D as the corresponding square 2D colour code only encodes two qubits.

This does not completely rule out more exotic slices in these codes (e.g. with logical qubits encoded in topological defects rather than code boundaries) but it seems unlikely that slices similar to the ones presented here exist for 3D colour codes.

It is worth emphasising the significant difference between the JIT decoding scheme we have examined in this work and Bombín's original JIT decoding proposal [Bombín, 2018a]. In particular, the above arguments regarding the difficulties of constructing valid colour code slices do not apply to that scheme because it uses an measurement-based formalism rather than a circuit-based one and so does not involve dimension jumping operations. These operations are what would map low-weight logicals in the slice to weight-$d$ logicals in the 2D code post-collapse, but the measurement-based formalism allows for a continuous "sliding" of the slice through the 3D spacetime and so dimension jumps are not required and these short error strings will be detected as we slide the slice past them. The JIT decoder proposed by Bombín in [Bombín, 2018a] also differs significantly from the one discussed here.

In light of recent results regarding transversal *CCCZ* gates in the 4D surface code [Kubica et al., 2015, Jochym-O'Connor and Yoder, 2021] it also seems natural to ask if an equivalent process could be used to construct slices of 4D surface code which allow for a linear-time *CCCZ* in the 3D surface code. We expect that such a generalisation should be possible, but note that its spacetime overhead would scale as $d^4$. This compares unfavourably with the $d^3$ scaling allowed by the constant-time set of computationally universal operations in the 3D surface code [Vasmer and Browne, 2019], and so is unlikely to be advantageous.

# Chapter 4

# Clifford Errors in 3D Topological Codes

The simulations of JIT decoding discussed in the previous chapter dealt only with $X$ errors and not with $Z$ errors. This was because the JIT decoder itself only supplied corrections for $X$ errors, but to establish a threshold for the full procedure we must also verify the performance of a decoder which addresses $Z$ errors in the code. The challenge of such a simulation is not the complexity of the decoder (which can be any valid decoder for pointlike syndromes in the 3D surface code, e.g. a 3D matching decoder) but of the $Z$ error distribution produced in the JIT decoding process. In addition to the random $Z$ errors resulting from noise in our system this distribution includes errors arising from the interaction of random $X$ errors and the non-Clifford $CCZ$ gate. This interaction produces $CZ$ errors which will be projected to distributions of $Z$ errors by our stabiliser measurements. These distributions are not random, and instead have a structure that depends on the stabiliser structure of the code.

More specifically, this interaction of a single-qubit $X$ error with the transversal $CCZ$ is

$$(CCZ)(X \otimes I \otimes I)(CCZ) = X \otimes CZ \tag{4.1}$$

and so an $X$ error in one code is mapped to a $CZ$ error in the other two codes by the application of the transversal $CCZ$. $CZ$ can be decomposed as $CZ = (I \otimes I + I \otimes Z +$

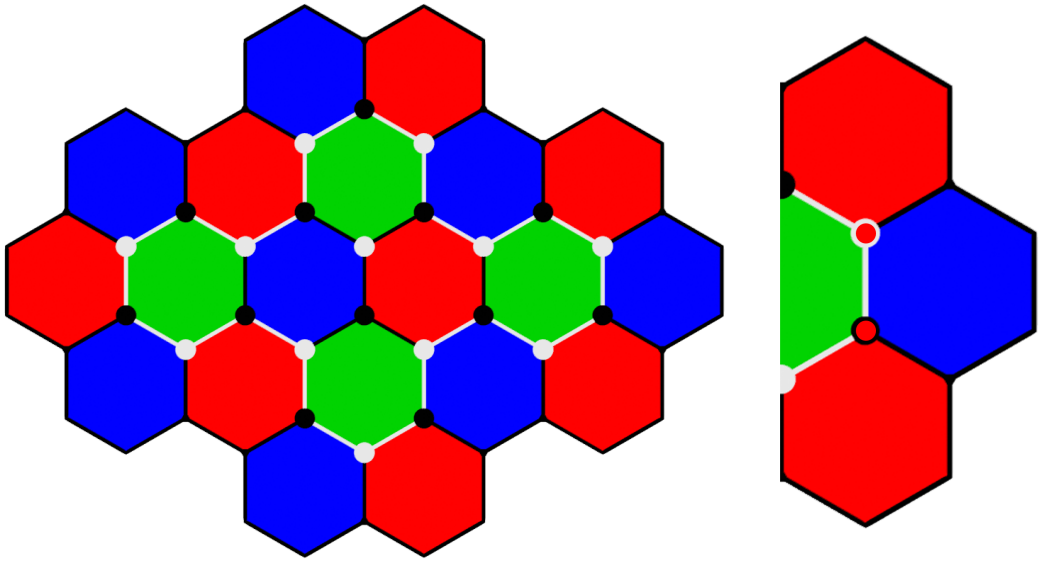
$Z \otimes I - Z \otimes Z)/2$ and so if a $CZ$ error occurs on a pair of qubits and then we were to make single-qubit measurements of the pair then we would expect to observe a $Z$ error on each with probability $p = 0.5$. It may therefore seem that to simulate these errors we need only randomly apply $Z$ errors to qubits in codes 2 and 3 whenever we have an $X$ error in code 1 (or any other combination) and for isolated single-qubit $X$ errors this is correct. However, for larger regions of $X$ errors the structure of the code introduces additional constraints on the kinds of error distributions we can obtain. These distributions have been studied previously for the cases of Clifford errors in the 2D and 3D colour codes [Yoshida, 2015, Bombín, 2018b] but (to our knowledge) relatively little work has been done on these errors apart from this. In this chapter we study the effects of these errors in 3D surface code, which differs from the colour code case as the non-Clifford gate in this setting is a multi-qubit entangling gate between several copies of the code, rather than a single-qubit gate that acts only on one code. Despite this we obtain similar results to these previous works, and in particular we show that the so-called "linking charge" phemonenon [Bombín, 2018b] observed in the case of the 3D colour code is not only replicated in the 3D surface code but has a much clearer origin in this setting.

We begin by reviewing the previous results on this topic and then discuss our results for the 3D surface code. Finally we apply our proof techniques to the case of the 3D colour code and show that they replicate the results of [Bombín, 2018b] while also highlighting some differences between this case and that of the 3D surface code.

## 4.1 Clifford Errors in the Colour Code

### 4.1.1 The 2D Colour Code

The problem of Clifford errors in the 2D colour code was examined in [Yoshida, 2015], although the author does not refer to them as such and instead considers these operators in the context of excitations in a symmetry-protected topological phase. Specifically, the author examines the effect of applying a pattern of alternating $S$ and $S^\dagger$ to all qubits within a particular region $\mathcal{R}$ defined by a subset of plaquettes of a particular colour as in fig. 4.1. The 2D colour code possesses a transveral $S$ gate



(a) (b)

**Figure 4.1:** A region of $S$ errors in the 2D colour code. $S$ is applied to all qubits marked with a white circle and $S^\dagger$ is applied to all qubits marked with a black circle. The qubits in this region are those on the vertices of the four green plaquettes inside the loop of white edges while the plaquettes which border this region are either red or blue. (b) A closeup of part of the boundary region of (a) and two $Z$ errors on qubits marked with red circles which anticommute with $X$ stabilisers on the two red plaquettes.

which can be implemented via such an application of $S$ and $S^\dagger$ to all qubits in the code and so this error can be thought of as a partial or incomplete logical operator. We can define the boundary of $\mathcal{R}$, $\partial\mathcal{R}$, to be the set of all plaquettes/stabiliser generators partially supported on $\mathcal{R}$. An important observation in this and all subsequent cases is that because logical operators must preserve the codespace (and therefore are not detectable by stabilisers of the code) this region of Clifford errors should only be detected by stabilisers in $\partial\mathcal{R}$. If it could be detected by a stabiliser not in $\partial\mathcal{R}$ then that stabiliser should also detect the logical $S$ gate as these operators are not locally distinguishable except on the boundary of $\mathcal{R}$.

The formally derived result of [Yoshida, 2015] agrees with this intuition. It says that if we apply the Clifford error shown in fig. 4.1, for example, and then measure the stabilisers of the code (specifically we only need to measure the $X$ stabilisers as $Z$ and $S$ commute) we will project the $S$ and $S^\dagger$ errors to a distribution of $Z$ errors

which anticommute only with the red and blue stabilisers in $\partial\mathcal{R}$, with each such stabiliser returning a $-1$ measurement outcome with probability $p = 0.5$. Recall that error strings in the 2D colour code must either anticommute with stabilisers of all three colours or anticommute with a pair of plaquettes of the same colour. The former is not possible in this case as there are only two colours of plaquette in $\partial\mathcal{R}$ and so the error distribution must be a collection of $Z$ strings supported on qubits of $\mathcal{R}$ which run between same-coloured plaquettes in $\partial\mathcal{R}$. Because each such string anticommutes with a pair of plaquettes the total number of $-1$ outcomes from plaquettes of each colour should be even. For example, fig. 4.1b shows a two-qubit $Z$ error which anticommutes with a pair of red plaquettes in $\partial\mathcal{R}$.

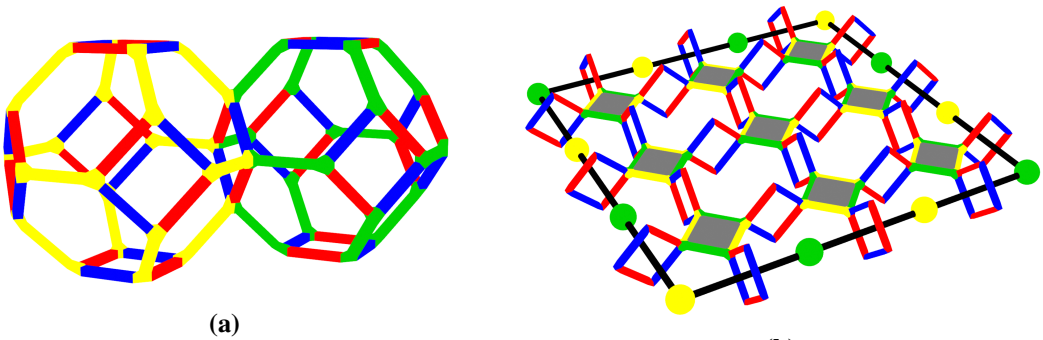
### 4.1.2 The 3D Colour Code

A corresponding analysis of $S$ errors in the 3D colour code can be found in [Bombín, 2018b]. This case is arguably of greater practical relevance as the 3D colour code admits a transversal $T$ gate implemented by an application of $T$ and $T^\dagger$ to a bicolouring of the vertices (qubits) of the lattice. This gate will create regions of $S$ and $S^\dagger$ errors wherever we have regions of $X$ errors as $TXT^\dagger = e^{-i\pi/4}SX$ and $T^\dagger XT = e^{i\pi/4}S^\dagger X$.

We can initially consider an $X$ error membrane defined on the vertices of a set of faces of colour $\kappa_1\kappa_2$ (by which we mean they are formed from edges of colour $\kappa_1$ and $\kappa_2$). This error will be detected by $Z$ stabiliser generators on faces of colour $\kappa_3\kappa_4$ at the boundary of the membrane. An example is shown in fig. 4.2b. When we apply the $\overline{T}$ gate described previously we will create $S$ errors wherever we apply $T$ and $S^\dagger$ errors wherever we apply $T^\dagger$.

Like the 2D colour code, the 3D colour code admits a transversal $S$ gate. This gate can be implemented by a membrane of $S$ and $S^\dagger$ (using the same colouring as the transversal $T$) with its edges at the boundaries of the code[1]. As with the 2D case, an $S$ error membrane such as the one in fig. 4.2b should only be detected by stabilisers at its boundary (i.e. by $X$ stabilisers on cells which the syndrome loop passes through) because it is only locally distinguishable from the logical $S$ operator

[1]This membrane of qubits also supports a logical $X$ operator. We can see that this configuration of $S$ and $S^\dagger$ should implement transversal $S$ because it will be created by applying transversal $T$ to a code in the logical $|1\rangle$ state, and in the logical space we should have $\overline{TXT}^\dagger = e^{-i\pi/4}\overline{SX}$



(a) (b)

**Figure 4.2:** (a) G (left) and Y (right) cells of a large 3D colour code. These cells meet at an RB face. (b) A membrane of $X$ errors in the 3D colour code. The error is supported on qubits on the vertices of YG faces (grey) and detected by $Z$ stabilisers on RB faces on the membrane's boundary. The resulting syndrome is shown by the black loop which passes through the centres of the violated $Z$ stabilisers. The two colours of dots on this syndrome mark the places where it passes through the centre of a G or Y cell. In order to improve visual clarity full cells are not shown.

in this region.

The results of [Bombín, 2018b] agree with [Yoshida, 2015] for this case, i.e. we expect measurement outcomes of $+1$ from all stabilisers except for $X$ stabilisers on the membrane boundary, which we expect to return random outcomes but with an even parity of $-1$s for each colour. However, more complex errors are possible in the 3D colour code and these are where the results diverge from the 2D case.

Consider a pair of intersecting membranes with linked syndromes as in fig. 4.3. This error is the product of two $X$ error membranes of the form discussed above, and so one might expect that application of transversal $T$ and measurement of the $X$ stabilisers on the boundaries of these membranes would once again give random outcomes with an even parity of violated stabilisers of each colour. However, what is shown in [Bombín, 2018b] is that we actually observe an odd number of violated stabilisers of each colour. This is consistent with a distribution of $Z$ errors as described previously plus an additional $Z$ error string running between the two membrane boundaries (such an error string anticommutes with a G and Y cell on one boundary and an R and a B cell on the other boundary). This is termed a "linking charge" of the two membranes, since in the topological phase perspective on the 3D colour code the charge distributions on the boundaries of the individual memrbranes

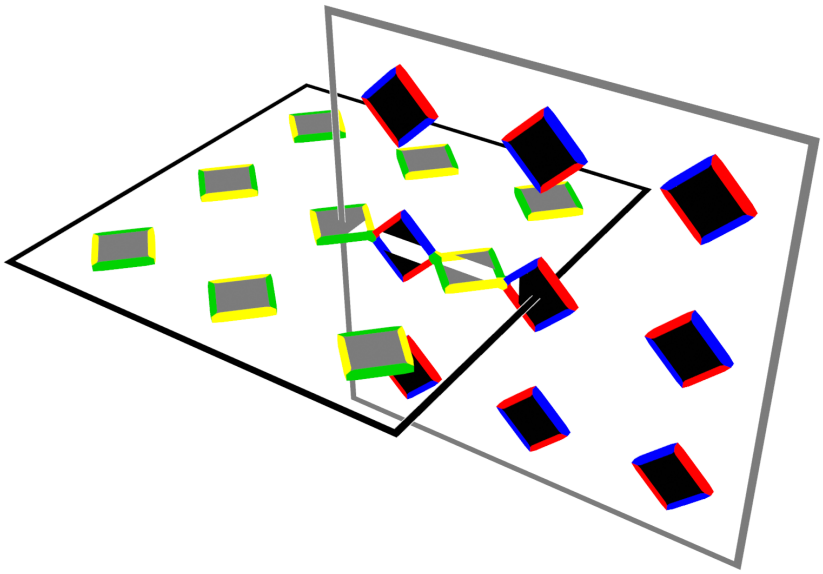


**Figure 4.3:** Two intersecting membranes of $X$ errors with linked syndromes in the 3D colour code. One is defined on YG faces and the other on RB faces. This error is the product of the two individual membranes so errors on the intersection cancel. One face of the YG membrane supports a $Z$ stabiliser which detects the RB membrane and vice versa. In order to improve visual clarity only relevant edges are shown.

are no longer independent. Previously each boundary was charge-neutral overall, whereas now the distribution for the pair of membranes is charge neutral but the distributions on individual membrane boundaries are not.

So far we have simply stated these results without providing a more in-depth explanation. We now provide a proof that these same phenomena occur in the 3D surface code, after which we will return to the case of the 3D colour code and show that our proof technique recovers these results in their original setting while also highlighting an interesting contrast with the surface code case.

## 4.2 Clifford Errors in the 3D Surface Code

### 4.2.1 Single Error Membrane in Cleanable Code Regions

In this section we will consider three 3D surface codes defined on a rectified lattice as in the previous chapter, and therefore admitting a transversal $CCZ$. We use notation where $X^c_\alpha$ implies $X$ operators on qubits from code $c \in \{1,2,3\}$ at vertices in the set $\alpha$. We start with a single a membranelike operator $X^1_\alpha$ detected by $Z$

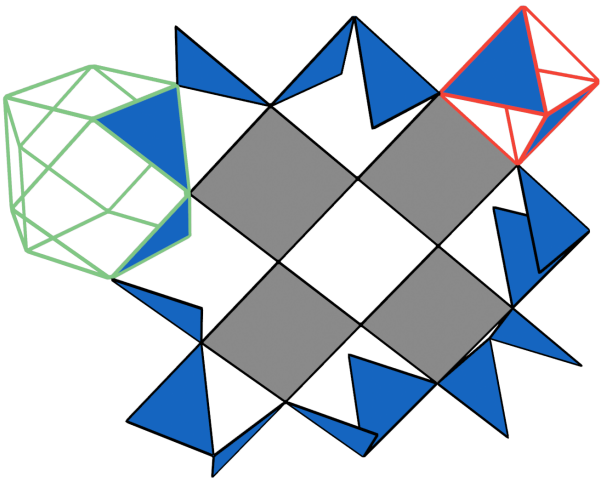


**Figure 4.4:** An $X$ error membrane in three copies of the 3D surface code using the rectified lattice. The error is supported on the qubits of the blue code which exist on vertices of the grey faces and is detected by $Z$ stabilisers on the blue faces. These faces are part of cells which support $X$ stabilisers of the red and green codes (two examples shown).

stabilisers of code 1 which are faces of cells in codes 2 and 3 as in fig. 4.4. We assume that this membrane exists in a cleanable region of the code, i.e. for any given logical Pauli operator of the code we can find an implementation of this operator which has trivial intersection with the membrane. Using the commutation relations $(CCZ)(X \otimes I \otimes I) = (X \otimes CZ)(CCZ)$ and $(CZ)(X \otimes I) = (X \otimes Z)(CZ)$ we see that applying transversal $CCZ$ in the presence of this error has the effect

$$\overline{CCZ} X^1_\alpha |\overline{\psi}\rangle = X^1_\alpha CZ^{23}_\alpha |\overline{\psi'}\rangle \tag{4.2}$$

where $|\overline{\psi}\rangle$ and $|\overline{\psi'}\rangle$ are states in the codespace of the three codes. $CZ^{23}_\alpha$ is a Clifford error analagous to the $S$ error membrane we observed in the 3D colour code. As with that error, this $CZ$ error becomes a logical operator if applied to the full support of a logical $X$ operator (as $\overline{CCZ}$ should preserve the codespace if $X^1_\alpha$ was a logical $X$ operator rather than an error). We therefore expect that, once again, this error should only be detected by stabilisers on the boundaries of the error membrane.

In order to consider the effect of $CZ^{23}_\alpha$ on the codestate $|\overline{\psi'}\rangle$ we can consider its effect individually on basis states. The state $|\overline{000}\rangle$ can be written as

$$|\overline{000}\rangle = \frac{1}{\sqrt{n}} \sum_{ijk} X^1_{\beta_i} X^2_{\beta_j} X^3_{\beta_k} |\mathbf{0}\rangle \tag{4.3}$$

where $X^m_\beta$ are $X$ stabilisers of code $m$ and $|\mathbf{0}\rangle$ is the all-zeros state of the qubits of all three codes. Other basis states can be written in a similar way by replacing these $X$ stabilisers with products of $X$ stabilisers and a logical $X$ operator for a given code. However, as we are currently considering an error membrane in a cleanable region of the code we can always choose these logical operators such that they have trivial intersection with the membrane and thus the analysis of any basis state is equivalent to the analysis for $|\overline{000}\rangle$ in this case. If we apply $CZ^{23}_\alpha$ to this state we find

$$CZ^{23}_\alpha |\overline{000}\rangle = \frac{1}{\sqrt{n}} \sum_{ijk} CZ^{23}_\alpha X^1_{\beta_i} X^2_{\beta_j} X^3_{\beta_k} |\mathbf{0}\rangle = \frac{1}{\sqrt{n}} \sum_{ijk} X^1_{\beta_i} X^2_{\beta_j} Z^3_{\alpha\cap\beta_j} X^3_{\beta_k} Z^2_{\alpha\cap\beta_k} |\mathbf{0}\rangle \tag{4.4}$$

where we have used that $CZ^{23}_\alpha$ acts trivially on $|\mathbf{0}\rangle$. We can then commute the $Z$ terms to the right and absorb them into $|\mathbf{0}\rangle$ to obtain

$$|\phi\rangle = CZ^{23}_\alpha |\overline{000}\rangle = \frac{1}{\sqrt{n}} \sum_{ijk} (-1)^{|\alpha\cap\beta_j\cap\beta_k|} X^1_{\beta_i} X^2_{\beta_j} X^3_{\beta_k} |\mathbf{0}\rangle . \tag{4.5}$$

In order to investigate the possible measurement outcomes of a specific stabiliser $X^m_{\beta_q}$ we can split the state into $|\phi\rangle = a|\phi^+_q\rangle + b|\phi^-_q\rangle$ where $X^m_{\beta_q}|\phi^+_q\rangle = |\phi^+_q\rangle$ and $X^m_{\beta_q}|\phi^-_q\rangle = -|\phi^-_q\rangle$. We consider only stabilisers from codes 2 and 3 as the $CZ$ error only has support on qubits from these two codes. Note that $|\phi^+_q\rangle$ must consist of pairs $X^l_{\beta_p}|\mathbf{0}\rangle + X^m_{\beta_q} X^l_{\beta_p}|\mathbf{0}\rangle$ whereas $|\phi^-_i\rangle$ must consist of pairs $X^l_{\beta_p}|\mathbf{0}\rangle - X^m_{\beta_q} X^l_{\beta_p}|\mathbf{0}\rangle$ (up to multiplication of the whole pair by $-1$). The former correspond to cases where $|\alpha\cap\beta_j\cap\beta_k|$ is even (so $(-1)^{|\alpha\cap\beta_j\cap\beta_k|} = 1$) while the latter correspond to cases where $|\alpha\cap\beta_j\cap\beta_k|$ is odd. We now consider three relevant types of stabiliser:

- Stabilisers not on membrane boundary: Recall that if $X^1_\alpha$ was a logical $X$ operator of code 1 then $CZ^{23}_\alpha$ should be a logical $CZ$ of codes 2 and 3 and so must preserve the stabiliser groups of these codes. This means that the intersection of a cell of code 2 (3) with the $CZ$ membrane must be the support

of a $Z$ stabiliser of code 3 (2) or $X$ stabilisers in one code would be mapped to $Z$ errors in the other. This means that $\alpha \cap \beta_j$ is the support of a $Z$ stabiliser of code 3 and since $\beta_k$ is the support of an $X$ stabiliser of code 3 $|\alpha \cap \beta_j \cap \beta_k|$ must be even for all such $\beta_j$ and $\beta_k$. Therefore $|\phi\rangle = |\phi_j^+\rangle = |\phi_k^+\rangle$ and we are in a $+1$ eigenstate of these stabilisers.

- Stabilisers generators (cells) on the membrane boundary: The error $X_\alpha^1$ is detected by $Z$ stabilisers supported on the faces of these cells as in fig. 4.4. Each such face is the intersection between a pair of generators $X_{\beta_j}^2$ and $X_{\beta_k}^3$ and so $|\alpha \cap \beta_j \cap \beta_k|$ must be odd or a $Z$ stabiliser on this face would not detect the error. There are two such faces for every generator on the membrane boundary (because the syndrome is a loop) and so every $X_{\beta_j}^2$ has two $X_{\beta_k}^3$ neighbours for which $|\alpha \cap \beta_j \cap \beta_k|$ is odd. These neighbours are disjoint and so $|\alpha \cap \beta_j \cap \beta_k|$ where $X_{\beta_k}^3$ is the product of these neighbours is even. Thus, for any stabiliser generator on the membrane boundary $X_{\beta_j}^2$ and any stabiliser $X_{\beta_k}^3$, $|\psi_j^+\rangle$ contains pairs $(X_{\beta_k}^3 + X_{\beta_j}^2 X_{\beta_k}^3)|\mathbf{0}\rangle$ where $X_{\beta_k}^3$ contains 0 or 2 neighbours of $X_{\beta_j}^2$ (on the boundary) while $|\psi_j^-\rangle$ contains pairs $(X_{\beta_k}^3 - X_{\beta_j}^2 X_{\beta_k}^3)|\mathbf{0}\rangle$ where $X_{\beta_k}^3$ contains only one of these neighbours. These pairs can also contain any $X_{\beta_i}^1$. There are equal numbers of each type of pair so $|\psi\rangle = \frac{1}{\sqrt{2}}(|\psi_j^+\rangle + |\psi_j^-\rangle)$ and we measure a random $\pm 1$ outcome from this stabiliser. An identical argument can be applied to generators of code 3.

- All stabilisers from one code on the membrane boundary: For any generator $X_{\beta_j}^2$ on the membrane boundary the intersection $|\alpha \cap \beta_j \cap \beta_k|$ with a neighbouring generator $X_{\beta_k}^3$ is odd (as discussed above). This means a $Z$ operator $Z_{\alpha \cap \beta_j}^3$ anticommutes with these $X_{\beta_k}^3$, and for each such $X_{\beta_k}^3$ there are two generators $X_{\beta_j}^2$ which have this property, so if $X_{\beta_j}^2$ is instead the product of all generators from code 2 on the membrane boundary then $Z_{\alpha \cap \beta_j}^3$ is a stabiliser. This means that $|\alpha \cap \beta_j \cap \beta_k|$ is even for all $X_{\beta_k}^3$ and so $|\psi\rangle = |\psi_j^+\rangle$ for this choice of $X_{\beta_j}^2$. Once again, an identical argument applies to $X_{\beta_k}^3$.

In summary this gives us an analagous result to what was observed for an

isolated membrane in the colour code. Stabilisers not on the boundary always give $+1$. Stabiliser generators on the boundary give $\pm 1$ randomly but we must get an even number of $-1$ stabilisers in any given code.

### 4.2.2 Linked Error Membranes in Cleanable Code Regions

Consider the case where we have $X$ error membranes in codes 1 and 2

$$\overline{CCZ} X^1_\alpha X^2_\gamma |\overline{\psi}\rangle \tag{4.6}$$

such that $\alpha \cap \gamma$ is nonemtpy. Then commuting the $CCZ$ to the right we have

$$X^1_\alpha CZ^{23}_\alpha X^2_\gamma CZ^{13}_\gamma |\overline{\psi'}\rangle = X^1_\alpha X^2_\gamma Z^3_{\alpha\cap\gamma} CZ^{23}_\alpha CZ^{13}_\gamma |\overline{\psi}\rangle \tag{4.7}$$

So we now have some $CZ$ errors acting on a codestate as before, but we also have a $Z$ string $Z^3_{\alpha\cap\gamma}$ on the intersection of these membranes in code 3. This is the surface code equivalent of the linking charge string described in [Bombín, 2018b]. Thus we expect that in code 1 we get random outcomes from stabilisers on the boundary of $CZ^{13}_\gamma$ and $+1$ outcomes from all others, with the total number of $-1$s even. In code 2 we expect the same thing on the boundary of $CZ^{23}_\alpha$. In code 3 we expect random outcomes from stabilisers on the boundaries of both membranes, and also expect an odd parity of $-1$ stabilisers on each boundary due to the linking charge string.

The simplicity of this statement stands in stark contrast to the complexity of the original proof of this phemonenon in the case of the colour code as presented in [Bombín, 2018b], to the extent that it may seem unremarkable to readers who are not familiar with that work. Additionally, it is not only the mathematical origin of linking charge that is clearer in the surface code but also its physical significance. The transversal $CCZ$ in this code works because the intersection of logical $X$ operators from any pair of codes is the support of a logical $Z$ operator in the third and so the logical action $\overline{CCZ}_{123}\overline{X}_1\overline{X}_2\overline{CCZ}_{123} = \overline{X}_1\overline{X}_2\overline{Z}_3\overline{CZ}_{23}\overline{CZ}_{13}$ is correctly implemented. Linking charge in this case is just another example of this creation of $Z$ strings on $X$ membrane intersections. This is consistent with claims made in [Bombín, 2018b]

that linking charge is important for the correct function of the transversal $T$ gate in the 3D colour code.

### 4.2.3 Error Membranes in Non-Cleanable Regions

Finally we address the case of error membranes which do not exist in cleanable regions of the code. Consider

$$\overline{CCZ}X^1_\alpha |\overline{\psi}\rangle = X^1_\alpha CZ^{23}_\alpha |\overline{\psi'}\rangle \tag{4.8}$$

where $CZ^{23}_\alpha$ now contains the support of a logical $Z$ operator in code 2 or 3. We choose code 2 but this choice is not important. This means that any implementation of logical $X$ in code 2 must have nontrivial intersection with this membrane and this changes our analysis for some codewords. For $|\overline{010}\rangle$ we have

$$\begin{aligned} CZ^{23}_\alpha |\overline{010}\rangle &= \frac{1}{\sqrt{n}} \sum_{ijk} CZ^{23}_\alpha X^1_{\beta_i} \overline{X}^2_L X^2_{\beta_j} X^3_{\beta_k} |\mathbf{0}\rangle \\ &= \frac{1}{\sqrt{n}} \sum_{ijk} X^1_{\beta_i} \overline{X}^2_L Z^3_{\alpha\cap L} X^2_{\beta_j} Z^3_{\alpha\cap\beta_j} X^3_{\beta_k} Z^2_{\alpha\cap\beta_k} |\mathbf{0}\rangle \\ &= \frac{Z^3_{\alpha\cap L}}{\sqrt{n}} \sum_{ijk} (-1)^{\alpha\cap\beta_j\cap\beta_k} X^1_{\beta_i} \overline{X}^2 X^2_{\beta_j} X^3_{\beta_k} |\mathbf{0}\rangle \end{aligned} \tag{4.9}$$

Where $\overline{X}^2_L$ is a logical $X$ implementation for code 2. We therefore create a global $Z$ error on the stringlike intersection of the error membrane support $\alpha$ and the logical operator support $L$. This does not affect observed syndromes (because $Z^3_{\alpha\cap L}$ runs from one side of the membrane to the other and so anticommutes with a pair of stabilisers on the membrane boundary) and also does not affect the encoded logical information unless $\alpha\cap L$ is the support of a logical $Z$ operator for code 3. Therefore we only get a logical error from applying $\overline{CZZ}$ to $X_\alpha$ if $\alpha$ contains the support of logical operators of both code 2 and code 3. Once again this contrasts with the case of the 3D colour code as in [Bombín, 2018b], where any $X$ error in a non-cleanable region results in a logical error after the application of the transversal non-Clifford.

In the next section we will apply the techniques used in this section to recover

the results of [Bombín, 2018b], highlighting the additional complexity present in the colour code in the process.

## 4.3 The 3D Colour Code Revisited

We now carry out the same analysis as above for the case of the 3D colour code. We recover the result of linking charge in this setting, although it requires more effort and is considerably less intuitive than the surface code case.

### 4.3.1 Single Error Membranes in Cleanable Code Regions

The colour code has a transversal $T$ gate corresponding to an application of $T$ and $T^\dagger$ to white and black vertices in a bicolouring of the lattice. Consider a membranelike error $X_\alpha$ (Pauli $X$ on all qubits in set $\alpha$ and identity otherwise) supported on a subset of faces of colour $\kappa_1\kappa_2$ and detected by $Z$ stabilisers on faces of colour $\kappa_3\kappa_4$ at the boundary of the membrane. Using that $TX = e^{-i\pi/4}SXT$ and $T^\dagger X = e^{i\pi/4}S^\dagger XT^\dagger$ we have that

$$\overline{T}X_\alpha\left|\overline{\psi}\right\rangle = e^{-i\pi N_w^\alpha/4}e^{i\pi N_b^\alpha/4}A_\alpha X_\alpha\left|\overline{\psi'}\right\rangle \tag{4.10}$$

where $N_w^\alpha$ and $N_b^\alpha$ are the numbers of white and black vertices in $\alpha$ and $A_\alpha$ is a tensor product of $S$ on all white vertices of $\alpha$ and $S^\dagger$ on all black vertices of $\alpha$. $\left|\overline{\psi}\right\rangle$ and $\left|\overline{\psi'}\right\rangle$ are states in the codespace. Using $SX = YS$, $S^\dagger X = -YS^\dagger$ and $Y = iXZ$ we have

$$e^{-i\pi N_w^\alpha/4}e^{i\pi N_b^\alpha/4}A_\alpha X_\alpha\left|\overline{\psi'}\right\rangle = e^{-i\pi N_w^\alpha/4}e^{i\pi N_b^\alpha/4}(-1)^{N_b^\alpha}i^{|\alpha|}X_\alpha Z_\alpha A_\alpha\left|\overline{\psi'}\right\rangle. \tag{4.11}$$

$\alpha$ is a product of faces of the code and faces are cycles in the lattice so must contain an equal number of $b$ and $w$ vertices so $e^{-i\pi N_w^\alpha/4}e^{i\pi N_b^\alpha/4} = 1$. If $|\alpha| = 0 \mod 4$ then $i^{|\alpha|} = (-1)^{N_b^\alpha} = 1$ and if $|\alpha| = 2 \mod 4$ then $i^{|\alpha|} = (-1)^{N_b^\alpha} = -1$. $Z_\alpha$ is a $Z$ stabiliser of the code (since it is a $Z$ operator supported on a set of faces) and commutes with $A_\alpha$ as they are both diagonal in the computational basis. In summary

$$\overline{T}X_\alpha \left|\overline{\psi}\right\rangle = X_\alpha A_\alpha \left|\overline{\psi'}\right\rangle. \tag{4.12}$$

We then want to know what effect $A_\alpha$ has on codestates. Since we are considering an error in a cleanable region of the code it is sufficient to consider only the effect on $|\overline{0}\rangle$ as the analysis for $|\overline{1}\rangle$ will be identical. We have that

$$|\overline{0}\rangle = \frac{1}{\sqrt{n}} \sum_{i=1}^{n} X_{\beta_i} |\mathbf{0}\rangle \tag{4.13}$$

where $X_{\beta_i}$ are stabilisers of the code (which are cells or products of cells of the lattice) and $|\mathbf{0}\rangle$ is the all-zeros state. We can then use the same commutation relations as above to show

$$A_\alpha \frac{1}{\sqrt{n}} \sum_{i=1}^{n} X_{\beta_i} |\mathbf{0}\rangle = \frac{1}{\sqrt{n}} \sum_{i=1}^{n} A_\alpha X_{\beta_i} |\mathbf{0}\rangle = \frac{1}{\sqrt{n}} \sum_{i=1}^{n} (-1)^{N_b^{\alpha\cap\beta_i}} i^{|\alpha\cap\beta_i|} X_{\beta_i} Z_{\alpha\cap\beta_i} |\mathbf{0}\rangle \tag{4.14}$$

$Z_{\alpha\cap\beta_i}$ acts trivially on the all-zeros state so we can rewrite this as

$$A_\alpha |\overline{0}\rangle = \frac{1}{\sqrt{n}} \sum_{i=1}^{n} i^{g(\alpha\cap\beta_i)} X_{\beta_i} |\mathbf{0}\rangle \tag{4.15}$$

where $g(\alpha\cap\beta_i) = |\alpha\cap\beta_i| + 2N_b^{\alpha\cap\beta_i}$. We can see that

$$\begin{aligned} (|\alpha\cap\beta_i| + 2N_b^{\alpha\cap\beta_i}) \mod 4 &= (N_w^{\alpha\cap\beta_i} + 3N_b^{\alpha\cap\beta_i}) \mod 4 \\ &= (N_w^{\alpha\cap\beta_i} - N_b^{\alpha\cap\beta_i}) \mod 4 \end{aligned} \tag{4.16}$$

so $g(\alpha\cap\beta_i)$ can be understood as the difference (mod 4) between the number of $b$ and $w$ vertices in $\alpha\cap\beta_i$. $\alpha$ is a set of faces of the code and the intersection of any face (or product of faces) with a cell (or product of cells) of the 3D colour code is even so $i^{g(\alpha\cap\beta_i)} = \pm 1$ and the effect of $A_\alpha$ on $|\overline{0}\rangle$ is to flip the sign of some of the terms in this superposition as in the surface code case. We can then once again write our state as

$$A_\alpha |\bar{0}\rangle = a|\phi_i^+\rangle + b|\phi_i^-\rangle \tag{4.17}$$

where $X_i|\phi_i^+\rangle = |\phi_i^+\rangle$ and $X_i|\phi_i^-\rangle = -|\phi_i^-\rangle$. The following lemma will be useful in finding values for $a$ and $b$.

**Lemma 4.0.1** *$i^{g(\alpha\cap(\beta_i+\beta_j))} \neq i^{g(\alpha\cap\beta_i)}i^{g(\alpha\cap\beta_j)}$ only if $|\alpha\cap\beta_i\cap\beta_j|$ is odd ($\beta_i+\beta_j$ is pointwise addition modulo 2)*

If $\beta_i$ and $\beta_j$ are disjoint then $\beta_i+\beta_j = \beta_i\cup\beta_j$ and so $\alpha\cap(\beta_i\cup\beta_j) = (\alpha\cap\beta_i)\cup(\alpha\cap\beta_j) = (\alpha\cap\beta_i)+(\alpha\cap\beta_j)$. This means

$$\begin{aligned} g(\alpha\cap(\beta_i+\beta_j)) &= |\alpha\cap(\beta_i+\beta_j)| + 2N_b^{\alpha\cap(\beta_i+\beta_j)} \\ &= |(\alpha\cap\beta_i)+(\alpha\cap\beta_j)| + 2N_b^{(\alpha\cap\beta_i)+(\alpha\cap\beta_j)} \\ &= |\alpha\cap\beta_i| + |\alpha\cap\beta_j| + 2N_b^{\alpha\cap\beta_i} + 2N_b^{\alpha\cap\beta_j} \\ &= g(\alpha\cap\beta_i) + g(\alpha\cap\beta_j) \end{aligned} \tag{4.18}$$

where we have used that $|a+b| = |a|+|b|$ for disjoint $a$ and $b$. Therefore we only have $i^{g(\alpha\cap(\beta_i+\beta_j))} \neq i^{g(\alpha\cap\beta_i)}i^{g(\alpha\cap\beta_j)}$ if $\beta_i\cap\beta_j$ is nonempty. In this case we can write $\beta_i\cup\beta_j = \beta_i'+\beta_j'+\beta_i\cap\beta_j$ where $\beta_i'$ ($\beta_j'$) are the elements of $\beta_i$ ($\beta_j$) not in $\beta_i\cap\beta_j$. $\beta_i+\beta_j = \beta_i'+\beta_j'$ and $\beta_i'$, $\beta_j'$ and $\beta_i\cap\beta_j$ are all disjoint, so by the same method as above we can show

$$g(\alpha\cap(\beta_i+\beta_j)) = g(\alpha\cap(\beta_i'+\beta_j')) = g(\alpha\cap\beta_i') + g(\alpha\cap\beta_j') \tag{4.19}$$

and

$$\begin{aligned} g(\alpha\cap\beta_i)+g(\alpha\cap\beta_j) &= g(\alpha\cap(\beta_i'+\beta_i\cap\beta_j)) + g(\alpha\cap(\beta_j'+\beta_i\cap\beta_j)) \\ &= g(\alpha\cap\beta_i') + g(\alpha\cap\beta_j') + 2g(\alpha\cap\beta_i\cap\beta_j) \end{aligned} \tag{4.20}$$

Thus for general $\beta_i$ and $\beta_j$ we have

$$i^{g(\alpha\cap\beta_i)}i^{g(\alpha\cap\beta_j)} = (-1)^{g(\alpha\cap\beta_i\cap\beta_j)}i^{g(\alpha\cap(\beta_i+\beta_j))} \tag{4.21}$$

and so $i^{g(\alpha\cap(\beta_i+\beta_j))} \neq i^{g(\alpha\cap\beta_i)}i^{g(\alpha\cap\beta_j)}$ only if $g(\alpha\cap\beta_i\cap\beta_j)$ is odd which means $|\alpha\cap\beta_i\cap\beta_j|$ is also odd. □

**Corollary 1:** If $Z_{\alpha\cap\beta_i}$ is a stabiliser then $i^{g(\alpha\cap(\beta_i+\beta_j))} = i^{g(\alpha\cap\beta_i)}i^{g(\alpha\cap\beta_j)}\ \forall\ \beta_j$. This is because the intersection of any $X$ and $Z$ stabiliser of the code must be even and so $\alpha\cap\beta_i\cap\beta_j$ must be even if $\alpha\cap\beta_i$ is the support of a $Z$ stabiliser.

Now let us reconsider the state

$$|\phi\rangle = A_\alpha|\overline{0}\rangle = \frac{1}{\sqrt{n}}\sum_{i=1}^{n} i^{g(\alpha\cap\beta_i)}X_i|\mathbf{0}\rangle = a|\phi_i^+\rangle + b|\phi_i^-\rangle \tag{4.22}$$

Note that once again $|\phi_i^+\rangle$ must consist of pairs $X_{\beta_j}|\mathbf{0}\rangle + X_{\beta_i}X_{\beta_j}|\mathbf{0}\rangle$ whereas $|\phi_i^-\rangle$ must consist of pairs $X_{\beta_j}|\mathbf{0}\rangle - X_{\beta_i}X_{\beta_j}|\mathbf{0}\rangle$ (up to multiplication of the whole pair by $-1$). These pairs are defined by $\beta_j$ and if $i^{g(\alpha\cap\beta_i)} = 1$ the pairs in $|\phi_i^-\rangle$ are all those for which $i^{g(\alpha\cap(\beta_i+\beta_j))} = -i^{g(\alpha\cap\beta_i)}i^{g(\alpha\cap\beta_j)}$, whereas if $i^{g(\alpha\cap\beta_i)} = -1$ then these pairs are in $|\phi_i^+\rangle$.

Now consider the same options for $X_i$ as in the surface code:

- Stabilisers not supported on the membrane boundary: The fact that $i^{g(\alpha\cap\beta_i)} = 1$ in this case is part of the requirement for transversal $T$ (see corollary 7 of [Rengaswamy et al., 2020]). Additionally, the intersection of these $X$ stabilisers with the membrane is either nothing or the support of a $Z$ stabiliser and so we always have $i^{g(\alpha\cap(\beta_i+\beta_j))} = i^{g(\alpha\cap\beta_i)}i^{g(\alpha\cap\beta_j)}$ for this choice of $X_i$ by corollary 1. This means that $|\phi\rangle = |\phi_i^+\rangle$ in this case, and so we are in a $+1$ eigenstate of these stabilisers.

- Stabiliser generators on the membrane boundary: Our error membrane is supported on faces of colour $\kappa_1\kappa_2$ and detected by faces of colour $\kappa_3\kappa_4$ which are the interfaces of $\kappa_1$ and $\kappa_2$ cells on the membrane boundary. A $\kappa_1$ cell meets the membrane at $\kappa_2$ coloured edges which each contain one $w$ and one

$b$ vertex and so the total numbers of each in $\alpha \cap \beta_i$ are equal and $i^{g(\alpha\cap\beta_i)} = 1$ for these stabilisers. The intersection of a $\kappa_1\kappa_2$ face, a $\kappa_1$ cell and a $\kappa_2$ cell is a single vertex (since the $\kappa_1$ cell meets the face at a $\kappa_2$ edge, the $\kappa_2$ cell meets the face at a $\kappa_1$ edge and these edges meet at a vertex) and therefore $|\alpha \cap \beta_i \cap \beta_j|$ is odd for a pair of neighbouring cells on the membrane boundary. Thus $|\phi_i^+\rangle$ is formed from pairs $(X_{\beta_j} + X_{\beta_i}X_{\beta_j})|\mathbf{0}\rangle$ where $X_{\beta_j}$ contains either 0 or 2 such neighbours of $X_{\beta_i}$ while $|\phi_i^-\rangle$ contains all pairs $(X_{\beta_j} - X_{\beta_i}X_{\beta_j})|\mathbf{0}\rangle$ where $X_{\beta_j}$ contains only one of the neighbours of $X_{\beta_i}$. There are equal numbers of each type of pair so $|\phi\rangle = \frac{1}{\sqrt{2}}(|\phi_i^+\rangle + |\phi_i^-\rangle)$ and we expect a random $\pm 1$ outcome from a measurement of $X_{\beta_i}$.

- All stabiliser generators of one colour on the membrane boundary: For any $\kappa_1$ cell $X_{\beta_i}$ and neighbouring $\kappa_2$ cell $X_{\beta_j}$ which are both on the membrane boundary we have that $|\alpha \cap \beta_i \cap \beta_j|$ is odd, and so $Z_{\alpha\cap\beta_i}$ is an error string which anticommutes with the two $\kappa_2$ coloured neighbours of $X_{\beta_i}$. If $X_{\beta_i}$ is instead the product of all $\kappa_1$ coloured cells on the membrane boundary then each $\kappa_2$ cell anticommutes individually with the string from each of its two $\kappa_1$ coloured neighbours and so commutes with their product. Thus $Z_{\alpha\cap\beta_i}$ is a stabiliser for this choice of $X_{\beta_i}$. Additionally, $i^{g(\alpha\cap\beta_i)} = 1$ since $i^g = 1$ for each cell individually and the cells are all disjoint. Therefore we have $|\phi\rangle = |\phi_i^+\rangle$ for this $X_{\beta_i}$ as well.

We have recoved the expected result for an isolated membrane: If we measure all stabiliser generators of the code then stabilisers not on the membrane boundary will return $+1$. Stabilisers on the membrane boundary return random $\pm 1$ outcomes, but the total parity of $-1$ stabilisers of any colour will always be even.

### 4.3.2 Linked Error Membranes in Cleanable Code Regions

Now we consider a pair of membranes of $X$ errors with one defined on $\kappa_1\kappa_2$ faces and detected by $\kappa_3\kappa_4$ faces and one defined on $\kappa_3\kappa_4$ faces and detected by $\kappa_1\kappa_2$ faces as in fig. 4.3. We will refer to these as $\alpha_{\kappa_1\kappa_2}$ and $\alpha_{\kappa_3\kappa_4}$ respectively. Following the application of $\overline{T}$ we have a state

$$\begin{aligned}\overline{T}X_{\alpha_{\kappa_1\kappa_2}}X_{\alpha_{\kappa_3\kappa_4}}|\overline{\psi}\rangle &= \overline{T}X_{\alpha_{\kappa_1\kappa_2}+\alpha_{\kappa_3\kappa_4}}|\overline{\psi}\rangle \\ &= X_{\alpha_{\kappa_1\kappa_2}+\alpha_{\kappa_3\kappa_4}}A_{\alpha_{\kappa_1\kappa_2}+\alpha_{\kappa_3\kappa_4}}|\overline{\psi'}\rangle\,.\end{aligned} \tag{4.23}$$

by the same reasoning as (4.12) and using the fact that the membranes are individually defined on the supports of $Z$ stabilisers (sets of faces) and so their product is also the support of a $Z$ stabiliser. Notice that, unlike in the surface code, we do not observe the emergence of a linking charge string at this point and in fact we observe no errors on the intersection at all since $\alpha_{\kappa_1\kappa_2}+\alpha_{\kappa_3\kappa_4} = \alpha_{\kappa_1\kappa_2}\cup\alpha_{\kappa_3\kappa_4} - \alpha_{\kappa_1\kappa_2}\cap\alpha_{\kappa_3\kappa_4}$. The linking charge string in this case will come from the action of the Clifford error on the codestate rather than directly from the commutation of the transversal non-Clifford through the original $X$ error.

Much of the analysis from above carries over to this case, and the only difference will be for $X_{\beta_i}$ partially supported on the intersection of the two membranes. Note that the same method used to prove (4.21) can equivalently be used to show

$$i^{g((\alpha_{\kappa_1\kappa_2}+\alpha_{\kappa_3\kappa_4})\cap\beta_i)} = (-1)^{g(\alpha_{\kappa_1\kappa_2}\cap\alpha_{\kappa_3\kappa_4}\cap\beta_i)}i^{g(\alpha_{\kappa_1\kappa_2}\cap\beta_i)}i^{g(\alpha_{\kappa_3\kappa_4}\cap\beta_i)} \tag{4.24}$$

Then we have that

- $X$ stabilisers at intersection endpoints: These stabilisers are on the boundary of one membrane and in the interior of the other and contain a single qubit in $\alpha_{\kappa_1\kappa_2}\cap\alpha_{\kappa_3\kappa_4}$. If this cell has colour $\kappa_1$ then it must meet the $\kappa_1\kappa_2$ membrane at a set of $\kappa_2$ coloured edges and the $\kappa_3\kappa_4$ membrane at a $\kappa_3\kappa_4$ coloured face. Sets of disjoint edges and individual faces both contain equal numbers of $b$ and $w$ vertices so $i^{g(\alpha_{\kappa_1\kappa_2}\cap\beta_i)} = i^{g(\alpha_{\kappa_3\kappa_4}\cap\beta_i)} = 1$. $\alpha_{\kappa_1\kappa_2}\cap\alpha_{\kappa_3\kappa_4}\cap\beta_i$ is a single qubit so $i^{g(\alpha\cap\beta_i)} = -1$ by (4.24).

- $X$ stabilisers on the intersection (not endpoints). $Z_{\alpha_{\kappa_1\kappa_2}\cap\beta_i}$ and $Z_{\alpha_{\kappa_3\kappa_4}\cap\beta_i}$ are both $Z$ stabilisers so $i^{g(\alpha_{\kappa_1\kappa_2}\cap\beta_i)} = i^{g(\alpha_{\kappa_3\kappa_4}\cap\beta_i)} = 1$. The product of two $Z$ stabilisers is another $Z$ stabiliser, and all $Z$ stabilisers have even weight so

$|(\alpha_{\kappa_1\kappa_2} \cap \beta_i) + (\alpha_{\kappa_3\kappa_4} \cap \beta_i)|$ is even and so $|\alpha_{\kappa_1\kappa_2} \cap \alpha_{\kappa_3\kappa_4} \cap \beta_i|$ is also even and $i^{g(\alpha\cap\beta_i)} = 1$.

For non-endpoint stabilisers everything is as before. For endpoint stabiliser generators we once again have $|\phi\rangle = \frac{1}{\sqrt{2}}(|\phi_i^+\rangle + |\phi_i^-\rangle)$ but we have swapped which pairs of states are in $|\phi_i^+\rangle$ and $|\phi_i^-\rangle$ since, e.g. $|\phi_i^-\rangle$ now contains the pair $|\mathbf{0}\rangle - X_{\beta_i}|\mathbf{0}\rangle$ whereas previously we had $|\mathbf{0}\rangle + X_{\beta_i}|\mathbf{0}\rangle$ in $|\phi_i^+\rangle$. If we consider all cells of one colour on one of the membrane boundaries then as before the intersection is a stabiliser and as before $i^{g(\alpha\cap\beta_i)}$ is the product of $i^g$ for all individual cells in the product as these cells are all disjoint. One of these cells sits at an intersection endpoint and so has $i^g = -1$ whereas the rest have $i^g = 1$ and so $i^{g(\alpha\cap\beta_i)} = -1$. Thus we now have $|\phi\rangle = |\phi_i^-\rangle$ whereas before we had $|\phi\rangle = |\phi_i^+\rangle$. This implies that when we measure all the stabiliser generators of the code we will once again get random outcomes from the membrane boundary stabilisers, but instead of having an even parity of each colour on each boundary we have an odd parity, and this is consistent with a linking charge string running between the two membrane boundaries.

### 4.3.3 Error Membranes in Non-Cleanable Regions

Consider an $X$ error membrane in the colour code which is supported on a subset of faces of colour $\kappa_1\kappa_2$ and also on the support of a logical $Z$ operator. As before we have

$$\overline{T}X_\alpha|\overline{\psi}\rangle = e^{-i\pi N_w^\alpha/4}e^{-i\pi N_b^\alpha/4}i^{g(\alpha)}X_\alpha Z_\alpha A_\alpha|\overline{\psi'}\rangle \tag{4.25}$$

$\alpha$ is a product of the supports of $Z$ stabilisers and a $Z$ logical. For the former $N_w = N_b$ and for the latter $N_w = N_b + 1$ so $e^{-i\pi N_w^\alpha/4}e^{-i\pi N_b^\alpha/4} = e^{-i\pi/4}$. Also $g(\alpha) = |\alpha| + 2N_b^\alpha = N_w^\alpha + N_b^\alpha + 2N_b^\alpha = 4N_b^\alpha + 1$ so $i^{g(\alpha)} = i$. This gives

$$\overline{T}X_\alpha|\overline{\psi}\rangle = e^{i\pi/4}X_\alpha Z_\alpha A_\alpha|\overline{\psi'}\rangle \tag{4.26}$$

Notice that $Z_\alpha$ is a logical $Z$ operator, whereas for cleanable $\alpha$ it was a stabiliser. Now we want to consider the effect of $A_\alpha$ on codestates. For $|\overline{0}\rangle$ the analysis is the same as before, but for $|\overline{1}\rangle$ we must now consider the interaction of logical $X$

operators with $A_\alpha$. $|\overline{1}\rangle$ can be written

$$|\overline{1}\rangle = \frac{1}{\sqrt{n}} \sum_{i=1}^{n} \overline{X} X_{\beta_i} |\mathbf{0}\rangle \tag{4.27}$$

where $\overline{X}$ is a logical $X$ operator. It does not matter which implementation of $\overline{X}$ we choose, so we choose it to be $X$ on all qubits. We then have that

$$A_\alpha |\overline{1}\rangle = \frac{1}{\sqrt{n}} \sum_{i=1}^{n} i^{g(\alpha)} \overline{X} Z_\alpha i^{g(\alpha \cap \beta_i)} X_{\beta_i} Z_{\alpha \cap \beta_i} |\mathbf{0}\rangle = \frac{i}{\sqrt{n}} \sum_{i=1}^{n} i^{g(\alpha \cap \beta_i)} \overline{X} X_{\beta_i} |\mathbf{0}\rangle \tag{4.28}$$

where we have used that $Z_\alpha$ is a logical $Z$ operator and so commutes with $X_{\beta_i}$ which is a stabiliser. Thus we see that the action of $A_\alpha$ on $|\overline{1}\rangle$ is the same as in the cleanable case except for a global factor of $i$. This is consistent with a logical $S$ error and so we conclude that in addition to creating distributions of $Z$ errors $A_\alpha$ also applies a logical $S$ to the codestate. Notice that in addition to this, commuting $\overline{T}$ through $X_\alpha$ created a logical $Z$ error $Z_\alpha$ in (4.26).

## 4.4 Discussion

We have generalised results regarding Clifford errors in the colour code to the case of the 3D surface code and found that not only do these results translate straightforwardly but they are much more easily understood in this setting. In the surface code the determininstic linking charge error string and the random error strings on the Clifford error boundary are separate processes, the former caused by commuting the transversal non-Clifford gate through a pair of Pauli errors and the latter caused by the action of a Clifford error on the codestate. In the colour code this distinction is less clear and these two kinds of $Z$ error have the same origin (the action of the the Clifford error on the codestate). The colour code and the three copies of the surface code are equivalent up to local unitary operations [Kubica, 2018] and so we propose that linking charge is best understood not as a colour code phenomenon but as a surface code one whose origin is obscured by the mapping to the colour code.

Linking charge in the surface code is more approachable not only from an analytic perspective but also from a numerical one. Simulating linking charge in the colour code requires a non-local method of detecting linked syndromes and as a result has been left out of previous numerical works [Beverland et al., 2021]. In contrast, simulating linking charge in the 3D surface code only requires us to apply $Z$ to any qubits of code $i$ where we have $X$ errors on the corresponding qubits of codes $j$ and $k$. This is a straightforward local check with minimal computational overhead and in fact the random sampling of error distributions on the membrane boundary constitutes the majority of the computational cost of this problem, meaning that a full simulation including linking charge in the surface code is as complex as a simulation which ignores it in the colour code.

Finally, we note that this result also has a straightforward generalisation to the case of the 4D surface code. This code admits a transversal $CCCZ$ between four copies of the code [Jochym-O'Connor and Yoder, 2021] and in the same way as (4.7) we have

$$\begin{aligned}\overline{CCCZ}X^1_\alpha X^2_\beta X^3_\gamma |\overline{\psi}\rangle &= X^1_\alpha CCZ^{234}_\alpha X^2_\beta CCZ^{134}_\beta X^3_\gamma CCZ^{124}_\gamma |\overline{\psi'}\rangle \\ &= X^1_\alpha X^2_\beta CZ^{34}_{\alpha\cap\beta} X^3_\gamma CZ^{24}_{\alpha\cap\gamma} CZ^{14}_{\beta\cap\gamma} |\psi''\rangle \\ &= X^1_\alpha X^2_\beta X^3_\gamma Z^4_{\alpha\cap\beta\cap\gamma} |\psi'''\rangle\end{aligned} \tag{4.29}$$

and so we obtain a $Z$ error string on the intersection of these three $X$ errors in addition to whatever effect the Clifford $CZ$ and non-Clifford $CCZ$ errors have on the codestate.

# Chapter 5

# General Conclusions

Quantum error correcting codes allow us to protect quantum information from environmental noise but reduce our ability to manipulate that information ourselves. Finding ways of circumventing this restriction and implementing a universal set of logical quantum gates is clearly essential for the development of a working quantum computer, and it is desirable to find gate sets that are as error-resistent and resource-efficient as possible. Transversal gates and magic state distillation are some of the most well known examples of such logical operations and in this thesis we have investigated several methods of performing quantum logic via the techniques of code deformation.

In chapter 2 we focused on performing logic with qubits encoded in twist defects. We generalised previous results for these defects in surface and colour codes to arbitrarily many copies of the surface code and classified the braiding and fusion relations for a subset of these general defects. We found that, as expected, the corresponding logical operations were all restricted to the Clifford group and that no new gates are obtained by generalising beyond the colour code (aside from tensor products of previously obtained gates). In chapter 3 we examined a proposal that uses a combination of code deformation and transversal gates, as well as an unusual decoding strategy, to implement a $CCZ$ gate in three copies of the 2D surface code. We carried out numerical simulations of part of this process (decoding $X$ errors) and obtained a threshold value of $\sim 0.1\%$ which represents both the first numerical evidence of a threshold for JIT decoding and a significant improvement

over the value obtained via analytical techniques. Additionally, we expect that this value could be improved via optimisation of the decoder. Finally, in chapter 4 we examined the problem of Clifford errors in the 3D surface code. This is relevant to the JIT decoding process discussed in chapter 3 and understanding of this problem is necessary for simulations of the $Z$ error decoding part of that procedure, but it also has more general relevance as these errors will also arise during the application of $CCZ$ to the full 3D surface code. We show that phenomena arising from Clifford errors in the 3D colour code are reproduced in the surface code case and also that these phenomena are more naturally understood in this setting.

One might ask what advantages these approaches to performing quantum logic have over techniques such as magic state distillation. For example, recent work has shown that a universal gate set achieved via dimension jumping in the colour code compares unfavourably with one achieved via state distillation and injection [Beverland et al., 2021] and so one might expect the same to be true for the JIT decoding scheme discussed here. While this is certainly possible, we do not believe it is guaranteed as the non-Clifford via JIT decoding requires fewer physical qubits and a simpler architecture than the full 3D code. It will also have a lower threshold, but we do not expect this threshold to be so low as to be experimentally unfeasable and so JIT decoding may be competitive with magic state distillation in this lower-noise regime.

Moving forward, we intend to continue this research and carry out simulations of the full JIT decoding procedure (including both $X$ and $Z$ errors using the results of chapter 4). We may wish to begin with simulations of Clifford errors in the full 3D surface code before adapting these simulations to the more complex JIT decoded case. Simulations of Clifford errors in 3D codes have been carried out previously in [Beverland et al., 2021] but these simulations did not include linking charge due to the difficulty of simulating that phenomenon in the colour code. The authors state that they do not expect linking charge to make a significant difference to the observed results, but due to the relative ease of simulting linking charge in the surface code we are now in a good position to investigate this claim and ascertain the exact impact

of linking charge error strings on the decoding process. As discussed in chapter 4, improved JIT decoding algorithms that utilise more of the information available in the syndrome history also present a promising avenue for future reasearch as this would also make JIT decoding schemes more competitive with other approaches to universal logic.

# Bibliography


Alan Agresti and Brent A. Coull. Approximate Is Better than "Exact" for Interval Estimation of Binomial Proportions. *The American Statistician*, 52(2):119–126, 1998. ISSN 0003-1305. doi: 10.2307/2685469. Publisher: [American Statistical Association, Taylor & Francis, Ltd.].

Arun B. Aloshious and Pradeep Kiran Sarvepalli. Decoding Toric Codes on Three Dimensional Simplical Complexes. *IEEE Transactions on Information Theory*, 67(2): 931–945, 2021. ISSN 1557-9654. doi: 10.1109/TIT.2020.3037042. Conference Name: IEEE Transactions on Information Theory.

Frank Arute et al. Quantum supremacy using a programmable superconducting processor. *Nature*, 574(7779):505–510, 2019. ISSN 1476-4687. doi: 10.1038/s41586-019-1666-5. Number: 7779 Publisher: Nature Publishing Group.

F. A. Bais and J. K. Slingerland. Condensate induced transitions between topologically ordered phases. *Physical Review B*, 79(4):045316, 2009. ISSN 1098-0121, 1550-235X. doi: 10.1103/PhysRevB.79.045316. arXiv: 0808.0627.

Maissam Barkeshli, Parsa Bonderson, Meng Cheng, and Zhenghan Wang. Symmetry, Defects, and Gauging of Topological Phases. *arXiv:1410.4540 [cond-mat, physics:hep-th, physics:math-ph, physics:quant-ph]*, 2014. arXiv: 1410.4540.

Michael E. Beverland, Aleksander Kubica, and Krysta M. Svore. The cost of universality: A comparative study of the overhead of state distillation and code switching with color codes. *arXiv:2101.02211 [quant-ph]*, 2021. arXiv: 2101.02211.

H. Bombín. Topological Order with a Twist: Ising Anyons from an Abelian Model. *Physical Review Letters*, 105(3):030403, 2010. doi: 10.1103/PhysRevLett.105.030403.

H. Bombín and M. A. Martin-Delgado. Topological Quantum Distillation. *Physical Review Letters*, 97(18):180501, 2006. ISSN 0031-9007, 1079-7114. doi: 10.1103/PhysRevLett.97.180501. arXiv: quant-ph/0605138.

H. Bombín and M. A. Martin-Delgado. Topological Computation without Braiding. *Physical Review Letters*, 98(16):160502, 2007. doi: 10.1103/PhysRevLett.98.160502. Publisher: American Physical Society.

Hector Bombín. 2D quantum computation with 3D topological codes. *arXiv:1810.09571 [quant-ph]*, 2018a. arXiv: 1810.09571.

Hector Bombín. Transversal gates and error propagation in 3D topological codes. *arXiv:1810.09575 [quant-ph]*, 2018b. arXiv: 1810.09575.

Héctor Bombín. Single-Shot Fault-Tolerant Quantum Error Correction. *Physical Review X*, 5(3):031043, 2015. doi: 10.1103/PhysRevX.5.031043. Publisher: American Physical Society.

Héctor Bombín. Dimensional jump in quantum error correction. *New Journal of Physics*, 18(4):043038, 2016. ISSN 1367-2630. doi: 10.1088/1367-2630/18/4/043038. Publisher: IOP Publishing.

J. Pablo Bonilla Ataides, David K. Tuckett, Stephen D. Bartlett, Steven T. Flammia, and Benjamin J. Brown. The XZZX surface code. *Nature Communications*, 12 (1):2172, 2021. ISSN 2041-1723. doi: 10.1038/s41467-021-22274-1.

S. B. Bravyi and A. Yu Kitaev. Quantum codes on a lattice with boundary. *arXiv:quant-ph/9811052*, 1998. arXiv: quant-ph/9811052.

Sergey Bravyi and Alexei Kitaev. Universal quantum computation with ideal Clifford gates and noisy ancillas. *Physical Review A*, 71(2):022316, 2005. doi: 10.1103/PhysRevA.71.022316. Publisher: American Physical Society.

Sergey Bravyi and Robert König. Classification of Topologically Protected Gates for Local Stabilizer Codes. *Physical Review Letters*, 110(17):170503, 2013. doi: 10.1103/PhysRevLett.110.170503. Publisher: American Physical Society.

Nikolas P. Breuckmann and Xiaotong Ni. Scalable Neural Network Decoders for Higher Dimensional Quantum Codes. *Quantum*, 2:68, 2018. doi: 10.22331/q-2018-05-24-68. Publisher: Verein zur Förderung des Open Access Publizierens in den Quantenwissenschaften.

Nikolas P. Breuckmann, Kasper Duivenvoorden, Dominik Michels, and Barbara M. Terhal. Local Decoders for the 2D and 4D Toric Code. *arXiv:1609.00510 [quant-ph]*, 2016. arXiv: 1609.00510.

Benjamin J. Brown. A fault-tolerant non-Clifford gate for the surface code in two dimensions. *Science Advances*, 6(21):eaay4929, 2020. ISSN 2375-2548. doi: 10.1126/sciadv.aay4929. Publisher: American Association for the Advancement of Science Section: Research Article.

Benjamin J. Brown, Katharina Laubscher, Markus S. Kesselring, and James R. Wootton. Poking Holes and Cutting Corners to Achieve Clifford Gates with the Surface Code. *Physical Review X*, 7(2):021029, 2017. doi: 10.1103/PhysRevX.7.021029.

Simon Burton and Dan Browne. Limitations on transversal gates for hypergraph product codes. *arXiv:2012.05842 [quant-ph]*, 2020. arXiv: 2012.05842.

A. R. Calderbank and Peter W. Shor. Good Quantum Error-Correcting Codes Exist. *Physical Review A*, 54(2):1098–1105, 1996. ISSN 1050-2947, 1094-1622. doi: 10.1103/PhysRevA.54.1098. arXiv: quant-ph/9512032.

Earl T. Campbell. A theory of single-shot error correction for adversarial noise. *Quantum Science and Technology*, 4(2):025006, 2019. ISSN 2058-9565. doi: 10.1088/2058-9565/aafc8f. Publisher: IOP Publishing.

Earl T. Campbell and Mark Howard. Unified framework for magic state distillation and multiqubit gate synthesis with reduced resource cost. *Physical Review A*, 95(2):022316, 2017. doi: 10.1103/PhysRevA.95.022316. Publisher: American Physical Society.

Earl T. Campbell, Barbara M. Terhal, and Christophe Vuillot. Roads towards fault-tolerant universal quantum computation. *Nature*, 549(7671):172–179, 2017. ISSN 1476-4687. doi: 10.1038/nature23460. Number: 7671 Publisher: Nature Publishing Group.

R. Craigen. Trace, Symmetry and Orthogonality. *Canadian Mathematical Bulletin*, 37(4):461–467, 1994. ISSN 0008-4395, 1496-4287. doi: 10.4153/CMB-1994-067-1.

Eric Dennis, Alexei Kitaev, Andrew Landahl, and John Preskill. Topological quantum memory. *Journal of Mathematical Physics*, 43(9):4452–4505, 2002. ISSN 0022-2488. doi: 10.1063/1.1499754. Publisher: American Institute of Physics.

K. Duivenvoorden, N. P. Breuckmann, and B. M. Terhal. Renormalization Group Decoder for a Four-Dimensional Toric Code. *IEEE Transactions on Information Theory*, 65(4):2545–2562, 2019. ISSN 1557-9654. doi: 10.1109/TIT.2018.2879937. Conference Name: IEEE Transactions on Information Theory.

Bryan Eastin and Emanuel Knill. Restrictions on Transversal Encoded Quantum Gate Sets. *Physical Review Letters*, 102(11):110502, 2009. doi: 10.1103/PhysRevLett.102.110502. Publisher: American Physical Society.

Jack Edmonds. Paths, Trees, and Flowers. *Canadian Journal of Mathematics*, 17:449–467, 1965. ISSN 0008-414X, 1496-4279. doi: 10.4153/CJM-1965-045-4. Publisher: Cambridge University Press.

Laird Egan, Dripto M. Debroy, Crystal Noel, Andrew Risinger, Daiwei Zhu, Debopriyo Biswas, Michael Newman, Muyuan Li, Kenneth R. Brown, Marko Cetina, and Christopher Monroe. Fault-Tolerant Operation of a Quantum Error-Correction Code. *arXiv:2009.11482 [quant-ph]*, 2021. arXiv: 2009.11482.

Emil Artin. *Galois theory: lectures delivered at the University of Notre Dame / by Dr. Emil Artin ; edited and supplemented with a section on applications by Dr. Arthur N. Milgram.* Notre Dame mathematical lectures ; no. 2. University of Notre Dame, University of Notre Dame Press, Notre Dame, Ind., Indiana, 2nd ed., with additions and revisions. edition, 1959.

Austin G. Fowler. Minimum weight perfect matching of fault-tolerant topological quantum error correction in average $O(1)$ parallel time. *arXiv:1307.1740 [quant-ph]*, 2014. arXiv: 1307.1740.

Austin G. Fowler, Matteo Mariantoni, John M. Martinis, and Andrew N. Cleland. Surface codes: Towards practical large-scale quantum computation. *Physical Review A*, 86(3):032324, 2012. doi: 10.1103/PhysRevA.86.032324.

Philippe Francesco, Pierre Mathieu, and David Sénéchal. *Conformal Field Theory*. Graduate Texts in Contemporary Physics. Springer-Verlag, New York, 1997. ISBN 978-0-387-94785-3.

Daniel Gottesman. Stabilizer Codes and Quantum Error Correction. *arXiv:quant-ph/9705052*, 1997. arXiv: quant-ph/9705052.

Lov K. Grover. A fast quantum mechanical algorithm for database search. In *Proceedings of the twenty-eighth annual ACM symposium on Theory of Computing*, STOC '96, pages 212–219, New York, NY, USA, 1996. Association for Computing Machinery. ISBN 978-0-89791-785-8. doi: 10.1145/237814.237866.

Jeongwan Haah, Matthew B. Hastings, D. Poulin, and D. Wecker. Magic state distillation with low space overhead and optimal asymptotic input count. *Quantum*, 1:31, 2017. doi: 10.22331/q-2017-10-03-31. Publisher: Verein zur Förderung des Open Access Publizierens in den Quantenwissenschaften.

A. Hedayat and W. D. Wallis. Hadamard Matrices and Their Applications. *The Annals of Statistics*, 6(6):1184–1238, 1978. ISSN 0090-5364, 2168-8966. doi: 10.1214/aos/1176344370.

Cornelius Hempel et al. Quantum Chemistry Calculations on a Trapped-Ion Quantum Simulator. *Physical Review X*, 8(3):031022, 2018. doi: 10.1103/PhysRevX.8.031022. Publisher: American Physical Society.

Clare Horsman, Austin G. Fowler, Simon Devitt, and Rodney Van Meter. Surface code quantum computing by lattice surgery. *New Journal of Physics*, 14(12): 123011, 2012. ISSN 1367-2630. doi: 10.1088/1367-2630/14/12/123011.

Adrian Hutter, James R. Wootton, and Daniel Loss. Parafermions in a Kagome Lattice of Qubits for Topological Quantum Computation. *Physical Review X*, 5 (4):041040, 2015. doi: 10.1103/PhysRevX.5.041040.

Tomas Jochym-O'Connor and Theodore J. Yoder. A four-dimensional toric code with non-Clifford transversal gates. *Physical Review Research*, 3(1):013118, 2021. ISSN 2643-1564. doi: 10.1103/PhysRevResearch.3.013118. arXiv: 2010.02238.

Tomas Jochym-O'Connor, Aleksander Kubica, and Theodore J. Yoder. Disjointness of Stabilizer Codes and Limitations on Fault-Tolerant Logical Gates. *Physical Review X*, 8(2):021047, 2018. doi: 10.1103/PhysRevX.8.021047. Publisher: American Physical Society.

Markus S. Kesselring, Fernando Pastawski, Jens Eisert, and Benjamin J. Brown. The boundaries and twist defects of the color code and their applications to topological quantum computation. *Quantum*, 2:101, 2018. doi: 10.22331/q-2018-10-19-101.

A. Yu. Kitaev. Fault-tolerant quantum computation by anyons. *Annals of Physics*, 303(1):2–30, 2003. ISSN 0003-4916. doi: 10.1016/S0003-4916(02)00018-0.

Alexei Kitaev. Anyons in an exactly solved model and beyond. *Annals of Physics*, 321(1):2–111, 2006. ISSN 0003-4916. doi: 10.1016/j.aop.2005.10.005.

Vladimir Kolmogorov. Blossom V: a new implementation of a minimum cost perfect matching algorithm. *Mathematical Programming Computation*, 1(1):43–67, 2009. ISSN 1867-2957. doi: 10.1007/s12532-009-0002-8.

Aleksander Kubica and Michael E. Beverland. Universal transversal gates with color codes: A simplified approach. *Physical Review A*, 91(3):032330, 2015. doi: 10.1103/PhysRevA.91.032330. Publisher: American Physical Society.

Aleksander Kubica and John Preskill. Cellular-Automaton Decoders with Provable Thresholds for Topological Codes. *Physical Review Letters*, 123(2):020501, 2019. doi: 10.1103/PhysRevLett.123.020501. Publisher: American Physical Society.

Aleksander Kubica, Beni Yoshida, and Fernando Pastawski. Unfolding the color code. *New Journal of Physics*, 17(8):083026, 2015. ISSN 1367-2630. doi: 10.1088/1367-2630/17/8/083026. arXiv: 1503.02065.

Aleksander Marek Kubica. *The ABCs of the Color Code: A Study of Topological Quantum Codes as Toy Models for Fault-Tolerant Quantum Computation and Quantum Phases Of Matter*. phd, California Institute of Technology, 2018.

Daniel Litinski. Magic State Distillation: Not as Costly as You Think. *Quantum*, 3: 205, 2019. doi: 10.22331/q-2019-12-02-205. Publisher: Verein zur Förderung des Open Access Publizierens in den Quantenwissenschaften.

Wilhelm Magnus, Abraham Karrass, and Donald Solitar. *Combinatorial Group Theory: Presentations of Groups in Terms of Generators and Relations*. Courier Corporation, 2004. ISBN 978-0-486-43830-6. Google-Books-ID: 1LW4s1RDRHQC.

Jonathan E. Moussa. Transversal Clifford gates on folded surface codes. *Physical Review A*, 94(4):042316, 2016. doi: 10.1103/PhysRevA.94.042316.

Gabriele Nebe, Eric M. Rains, and Neil J. A. Sloane. *Self-Dual Codes and Invariant Theory*. Algorithms and Computation in Mathematics. Springer-Verlag, Berlin Heidelberg, 2006. ISBN 978-3-540-30729-7. doi: 10.1007/3-540-30731-1.

P. J. J. O'Malley et al. Scalable Quantum Simulation of Molecular Energies. *Physical Review X*, 6(3):031007, 2016. doi: 10.1103/PhysRevX.6.031007. Publisher: American Physical Society.

Jiannis K. Pachos. Introduction to Topological Quantum Computation by Jiannis K. Pachos, 2012.

Pavel Panteleev and Gleb Kalachev. Degenerate Quantum LDPC Codes With Good Finite Length Performance. *arXiv:1904.02703 [quant-ph]*, 2019. arXiv: 1904.02703.

Armanda O. Quintavalle, Michael Vasmer, Joschka Roffe, and Earl T. Campbell. Single-shot error correction of three-dimensional homological product codes. *PRX Quantum*, 2(2):020340, 2021. ISSN 2691-3399. doi: 10.1103/PRXQuantum.2.020340. arXiv: 2009.11790.

Robert Raussendorf and Jim Harrington. Fault-Tolerant Quantum Computation with High Threshold in Two Dimensions. *Physical Review Letters*, 98(19):190504, 2007. doi: 10.1103/PhysRevLett.98.190504. Publisher: American Physical Society.

Robert Raussendorf, Sergey Bravyi, and Jim Harrington. Long-range quantum entanglement in noisy cluster states. *Physical Review A*, 71(6):062313, 2005. doi: 10.1103/PhysRevA.71.062313. Publisher: American Physical Society.

Narayanan Rengaswamy, Robert Calderbank, Michael Newman, and Henry D. Pfister. On Optimality of CSS Codes for Transversal $T$. *arXiv:1910.09333 [quant-ph]*, 2020. arXiv: 1910.09333.

Joschka Roffe, David R. White, Simon Burton, and Earl T. Campbell. Decoding Across the Quantum LDPC Code Landscape. *Physical Review Research*, 2(4):043423, 2020. ISSN 2643-1564. doi: 10.1103/PhysRevResearch.2.043423. arXiv: 2005.07016.

Jerzy Rozanski. Bicharacters, braids and Jacobi identity. *arXiv:q-alg/9611029*, 1996. arXiv: q-alg/9611029.

T. R. Scruby and D. E. Browne. A Hierarchy of Anyon Models Realised by Twists in Stacked Surface Codes. *Quantum*, 4:251, 2020. doi: 10.22331/q-2020-04-06-251.

Publisher: Verein zur Förderung des Open Access Publizierens in den Quantenwissenschaften.

T. R. Scruby, D. E. Browne, P. Webster, and M. Vasmer. Numerical Implementation of Just-In-Time Decoding in Novel Lattice Slices Through the Three-Dimensional Surface Code. *arXiv:2012.08536 [quant-ph]*, 2021. arXiv: 2012.08536.

Peter W. Shor. Scheme for reducing decoherence in quantum computer memory. *Physical Review A*, 52(4):R2493–R2496, 1995. doi: 10.1103/PhysRevA.52.R2493. Publisher: American Physical Society.

P.W. Shor. Algorithms for quantum computation: discrete logarithms and factoring. In *Proceedings 35th Annual Symposium on Foundations of Computer Science*, pages 124–134, 1994. doi: 10.1109/SFCS.1994.365700.

Jacob A. Siehler. Braided Near-group Categories. *arXiv:math/0011037*, 2000. arXiv: math/0011037.

A. M. Steane. Error Correcting Codes in Quantum Theory. *Physical Review Letters*, 77(5):793–797, 1996a. doi: 10.1103/PhysRevLett.77.793. Publisher: American Physical Society.

Andrew Steane. Multiple Particle Interference and Quantum Error Correction. *Proceedings of the Royal Society of London. Series A: Mathematical, Physical and Engineering Sciences*, 452(1954):2551–2577, 1996b. ISSN 1364-5021, 1471-2946. doi: 10.1098/rspa.1996.0136. arXiv: quant-ph/9601029.

Ashley M. Stephens. Fault-tolerant thresholds for quantum error correction with the surface code. *Physical Review A*, 89(2):022321, 2014. doi: 10.1103/PhysRevA.89.022321. Publisher: American Physical Society.

J. J. Sylvester. LX. Thoughts on inverse orthogonal matrices, simultaneous sign-successions, and tessellated pavements in two or more colours, with applications to Newton's rule, ornamental tile-work, and the theory of numbers. *The London,*

*Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, 34(232): 461–475, 1867. ISSN 1941-5982. doi: 10.1080/14786446708639914.

Daisuke Tambara and Shigeru Yamagami. Tensor Categories with Fusion Rules of Self-Duality for Finite Abelian Groups. *Journal of Algebra*, 209(2):692–707, 1998. ISSN 0021-8693. doi: 10.1006/jabr.1998.7558.

Michael Vasmer and Dan E. Browne. Three-dimensional surface codes: Transversal gates and fault-tolerant architectures. *Physical Review A*, 100(1):012312, 2019. doi: 10.1103/PhysRevA.100.012312. Publisher: American Physical Society.

Michael Vasmer, Dan E. Browne, and Aleksander Kubica. Cellular automaton decoders for topological quantum codes with noisy measurements and beyond. *Scientific Reports*, 11(1):2027, 2021. ISSN 2045-2322. doi: 10.1038/s41598-021-81138-2. arXiv: 2004.07247.

Christophe Vuillot, Lingling Lao, Ben Criger, Carmen García Almudéver, Koen Bertels, and Barbara M. Terhal. Code deformation and lattice surgery are gauge fixing. *New Journal of Physics*, 21(3):033028, 2019. ISSN 1367-2630. doi: 10.1088/1367-2630/ab0199. Publisher: IOP Publishing.

David S. Wang, Austin G. Fowler, and Lloyd C. L. Hollenberg. Surface code quantum computing with error rates over 1%. *Physical Review A*, 83(2):020302, 2011. doi: 10.1103/PhysRevA.83.020302. Publisher: American Physical Society.

Paul Webster and Stephen D. Bartlett. Braiding defects in topological stabiliser codes of any dimension cannot be universal. *arXiv:1811.11789 [quant-ph]*, 2018a. arXiv: 1811.11789.

Paul Webster and Stephen D. Bartlett. Locality-preserving logical operators in topological stabilizer codes. *Physical Review A*, 97(1):012330, 2018b. doi: 10.1103/PhysRevA.97.012330.

Paul Webster and Stephen D. Bartlett. Fault-Tolerant Quantum Gates with Defects

in Topological Stabiliser Codes. *arXiv:1906.01045 [quant-ph]*, 2019. arXiv: 1906.01045.

Paul Webster and Stephen D. Bartlett. Fault-tolerant quantum gates with defects in topological stabilizer codes. *Physical Review A*, 102(2):022403, 2020. doi: 10.1103/PhysRevA.102.022403. Publisher: American Physical Society.

Paul Webster, Michael Vasmer, Thomas R. Scruby, and Stephen D. Bartlett. Universal Fault-Tolerant Quantum Computing with Stabiliser Codes. *arXiv:2012.05260 [quant-ph]*, 2021. arXiv: 2012.05260.

Beni Yoshida. Topological color code and symmetry-protected topological phases. *Physical Review B*, 91(24):245131, 2015. doi: 10.1103/PhysRevB.91.245131. Publisher: American Physical Society.

Bei Zeng, Andrew Cross, and Isaac L. Chuang. Transversality Versus Universality for Additive Quantum Codes. *IEEE Transactions on Information Theory*, 57(9): 6272–6284, 2011. ISSN 1557-9654. doi: 10.1109/TIT.2011.2161917. Conference Name: IEEE Transactions on Information Theory.