University College London

Department of Physics and Astronomy

Fault-tolerant quantum computing with three-dimensional surface codes

Author:
Michael John George Vasmer

Supervisor:
Professor Dan Browne

Submitted in partial fulfilment for the degree of **Doctor of Philosophy**

December 5, 2019

I, Michael John George Vasmer, 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 thesis.

Signature

December 5, 2019

Date

Abstract

Quantum computers are far more error-prone than their classical counterparts. Therefore, to build a quantum computer capable of running large-scale quantum algorithms, we must use the techniques of quantum error correction to ensure that the computer produces the correct output even when its components are unreliable. However, the resource requirements of building such a fault-tolerant quantum computer are currently prohibitive. Here, we examine the utility of using three-dimensional (3D) surface codes in a fault-tolerant quantum computer. This family of topological error-correcting codes is a generalization of the well-known 2D surface code to three spatial dimensions.

We show that certain 3D surface codes have a transversal logical non-Clifford gate. In a quantum computing architecture, a non-Clifford gate is required to achieve computational universality. Transversal gates do not entangle qubits in different codes, so they are naturally fault tolerant because they do not spread errors. Next, we consider the problem of decoding 3D surface codes. In a quantum error-correcting code, we cannot observe the qubits directly, so we measure parity-check operators to gain information about the state of the code. Decoding is the problem of estimating what error has occurred given a list of unsatisfied parity checks. We observe that 3D surface codes offer asymmetric protection against bit-flip and phase-flip errors, but in both cases, we find that a threshold error rate exists below which we can suppress logical errors by increasing the size of the code.

We use our results about logical gates and decoding to propose two fault-tolerant quantum computing architectures that utilize 3D surface codes. Finally, we compare the resource requirements of our architectures with the requirements of leading quantum computing architectures based on topological codes. We find that one of our architectures may be competitive with the leading architectures, depending on the properties of the physical systems used to build the qubits.

Impact Statement

It is widely accepted that quantum computers have the potential to have a large commercial and societal impact. A fully fault-tolerant quantum computer would be able to perform calculations that are impossible to do using regular computers. For example, with a fault-tolerant quantum computer, we could break Rivest–Shamir–Adleman (RSA) encryption and simulate complex chemical reactions exponentially faster than is currently possible. The work in this thesis does not deal directly with the applications of quantum computers, however our results could help to make fault-tolerant quantum computers a reality sooner.

This thesis could benefit groups of researchers and engineers who are building small prototype quantum computers. There are many such groups working in academia, in start-ups, and in large corporations all across the world. At some point in the next five to ten years, we anticipate that one of these groups will build a quantum computer with enough qubits to realize error-correcting codes such as the three-dimensional (3D) surface codes we study in this thesis. When this happens, our results may inform the design and operation of such a quantum computer and future quantum computers.

Our results may also have additional impact, because the techniques we develop in this thesis could be applied to quantum error-correcting codes apart from the 3D surface code. For example, we anticipate that the cellular automaton decoder we developed for 3D surface codes could be used to decode other quantum error-correcting codes. We expect our methods to be of interest to researchers working in the field of quantum error correction, as well as people working for companies who are attempting to build large-scale quantum computers.

Acknowledgements

First of all, I would like to thank my supervisor, Dan Browne, for his support and encouragement during my PhD studies. I have learned an enormous amount from him about doing science and communicating my work effectively. I am also indebted to the other members of Dan's research group for giving me valuable advice and support. I especially want to thank my office-mate, Padraic Calpin, for sharing the ups and downs of PhD life with me, and Niko Breuckmann for many interesting and useful conversations. I am grateful to have been able to meet and talk to academics visiting UCL, especially Tom Stace and Paul Webster, who both helped me to sharpen my thinking on multiple occasions.

Being part of the Quantum Technologies CDT has enhanced my experience at UCL immeasurably, and I want to thank cohort two in particular — Alex, Dan, David, Gavin, James, Nathanaël, Padraic, Paul, Simon and Sofia — for the good times. I am especially thankful for Lopa's support and unique sense of humour. I have also been lucky to get the chance to collaborate with and visit many wonderful academics from outside UCL. In particular, I want to thank Alex Kubica, Earl Campbell and Joschka Roffe for their hospitality and for illuminating discussions.

I couldn't have finished my PhD without the support of my family and friends. I am grateful to my parents, David and Rachael, for their support and encouragement. I want to thank my grandparents for their support, and I am particularly grateful to my Grandad, John, for his insightful questions about my work. I am thankful to have had the support of my girlfriend, Hannah, throughout my PhD. I want to thank her in particular for proof-reading so much of my work. Finally, I want to thank all of my friends in London and North Wales, who have made my life outside my PhD so enjoyable.

Contents

1 Introduction **19**

1.1 Quantum information and quantum errors 20

1.2 Protecting quantum information 22

1.2.1 Surface codes 24

1.2.2 Subsystem codes 32

1.3 Processing encoded quantum information 33

1.3.1 Fault-tolerance with 2D surface codes 34

1.3.2 Restrictions on non-Clifford gates in topological codes 40

1.3.3 Colour codes 42

2 Transversal gates in three-dimensional surface codes **47**

2.1 The rectified picture 47

2.1.1 The rotated picture 48

2.1.2 Rectifying 3D surface codes 51

2.1.3 A family of 3D surface codes with boundaries 57

2.2 Transversal CCZ 66

2.2.1 Transversal CZ 73

2.2.2 Universal gate set 73

2.3 Generalizations of the rectified picture 74

2.3.1 Transforming surface codes into colour codes 77

2.3.2 Codes from Coxeter diagrams 81

3 Decoding three-dimensional surface codes **89**

3.1 Z-error decoding 90

3.2 X-error decoding 94

3.2.1 Sweep Rule with boundaries 98

3.2.2 Proof of error threshold 105

3.2.3 Numerical error threshold estimates 117

4 Fault-tolerant three-dimensional surface code architectures 127

4.1 Fault-tolerance in a single stack 127

4.2 3D surface code lattice surgery 130

4.2.1 3D/3D lattice surgery 131

4.2.2 2D/3D lattice surgery 133

4.3 Architectures and overheads 135

4.3.1 Purely 3D architecture 136

4.3.2 Hybrid 2D/3D architecture 139

5 Conclusion 143

Appendices 147

A Proof of Lemma 1 147

Bibliography 149

List of Figures

1.1 The stabilizer generators and boundaries of a 2D surface code. . . . 25
1.2 Primal and dual 2D surface code lattices. 26
1.3 Errors in 2D surface codes. 27
1.4 The error threshold of the 2D surface code. 28
1.5 Error syndromes in 3D surface codes. 30
1.6 A stabilizer measurement circuit. 34
1.7 A T-state injection circuit. 39
1.8 The regular 2D colour code lattices. 43

2.1 Truncating a cube. 48
2.2 2D surface codes in the rotated picture. 49
2.3 A 3D surface code shown in the Kitaev picture and the rectified picture. 52
2.4 Polyhedra and their duals. 52
2.5 The rectified cubic lattice. 53
2.6 The rhombic-dodecahedral lattice. 54
2.7 Arranging three lattices so that their rectifications are identical. . . 55
2.8 The stabilizer generators of the codes in a stack of 3D surface codes. 56
2.9 The $d = 3$ rectified cubic lattice with boundaries. 58
2.10 The two types of layer in a rectified cubic lattice. 58
2.11 Stabilizers on the boundaries of a rectified cubic lattice. 60
2.12 Redundant Z-stabilizers on the boundaries of a rectified cubic lattice. 64
2.13 Canonical logical operators in a stack of 3D surface codes. 65
2.14 Overlapping stabilizers on the boundaries of a rectified cubic lattice. 72
2.15 Implementing a Hadamard gate via measurement. 74
2.16 Four-colouring a cubic lattice. 76
2.17 The [[8,3,2]] colour code. 79
2.18 Applying the [[8,3,2]] transformation to a single vertex. 80

2.19 Applying the [[8,3,2]] transformation to a stack of 3D surface codes. 80
2.20 Generating tessellations using Wythoff's construction. 82
2.21 Topological code lattices with the same symmetry. 83
2.22 The fundamental region of a 3D Coxeter group. 84
2.23 A tesseract projected into 3D space. 86

3.1 The phase-flip error threshold of the 3D surface code. 93
3.2 The dynamics of Toom's rule. 95
3.3 The phase diagram of Toom's rule. 97
3.4 Errors that are not removed by the Sweep Rule decoder. 98
3.5 The future and past of a subset of vertices. 101
3.6 A causal region in a simplicial lattice with boundaries. 101
3.7 An illustration of the Sweep Rule cellular automaton update rule. . . 104
3.8 3D surface code lattices in the Kitaev picture. 118
3.9 Bit-flip error thresholds of 3D surface codes. 119
3.10 Errors that cause the SweepRule decoder to fail. 120
3.11 The sustainable bit-flip error threshold of 3D surface codes. 123
3.12 Optimization of the SweepRule decoder. 124

4.1 A rectified cubic lattice that supports three 3D surface codes. 128
4.2 Implementing a Hadamard gate via measurement. 130
4.3 Lattice surgery in 3D surface codes. 132
4.4 Lattice surgery between 3D and 2D surface codes. 134
4.5 Transferring quantum information using a $Z \otimes Z$ measurement. . . . 135
4.6 Implementing a $CNOT$ gate using $Z \otimes Z$ and $X \otimes X$ measurements. 135
4.7 3D surface code architecture layout. 137
4.8 A CCZ state injection circuit. 139
4.9 Dimensional jumps in 3D surface codes. 141

List of Tables

2.1 The stabilizer generators of a stack of three surface codes. 56

2.2 The stabilizer generators of a stack of four 3D surface codes. 75

2.3 Using Wythoff's construction to generate related 4D lattices. 85

3.1 Comparison of the SweepRule decoder with other decoders. 124

List of Abbreviations and Symbols

$\mathbb{C}$	The set of all complex numbers
$\mathbb{F}_2$	The finite field with two elements
$\mathbb{Z}$	The set of all integers
0D	Zero-dimensional
1D	One-dimensional
2D	Two-dimensional
3D	Three-dimensional
4D	Four-dimensional
bcc	Body-centred cubic
CA	Cellular automaton
CCZ	Controlled-controlled-Z gate
CNOT	Controlled-NOT gate
CPU	Central processing unit
CSS	Calderbank-Shor-Steane
CZ	Controlled-Z gate
JIT	Just-in-time
LPLO	Locality-preserving logical operator
MSD	Magic state distillation
MWPM	Minimum-weight perfect matching
RG	Renormalization group

Chapter 1

Introduction

In recent years, there has been an upsurge of investment in quantum computing by large companies and start-up funders. In the near term, quantum computers are likely to be relatively unreliable because of the difficulty of constructing many interacting quantum bits (qubits) that are simultaneously well-isolated from their environment and controllable with high precision. Early quantum computers will undoubtedly enable interesting research, but there is no guarantee that near-term quantum computers will have a substantial commercial impact. To run quantum algorithms that are both exponentially faster than classical algorithms (e.g. [1, 2]) and have clear commercial applications (e.g. [3]), we need a fault-tolerant quantum computer. Such a quantum computer would be still produce the correct output even when some of its components fail. But there is a price to pay for fault tolerance. Recently, various companies have claimed to posses quantum computers with 50–128 qubits [4, 5, 6]. However, using the most efficient architectures, it is estimated that hundreds of thousands of qubits would be required to build a fault-tolerant quantum computer capable of running large-scale quantum algorithms with exponential speed-up [3, 7]. Therefore, to make fault-tolerant quantum computing a reality, we must build more reliable qubits and increase the efficiency of fault-tolerant protocols. In this thesis, we focus on the second of these tasks.

One of the current leading proposals for fault-tolerant quantum computing is based on a quantum error-correcting code called the two-dimensional (2D) surface code [8, 9, 10]. This code has a high error threshold [11, 12, 13, 14] which makes it a very good candidate for storing quantum information. However, processing information using the 2D surface code is more challenging [15]. A resource-intensive procedure known as magic state distillation (MSD) is required to do universal quantum computation using 2D surface codes. Therefore, in order to reduce the resources required to build a fault-tolerant quantum computer, one option is to consider using other codes where processing quantum information is easier. In this thesis, we present results concerning one such code, the 3D surface code (the generalization of the 2D surface code to three spatial dimensions). We show that MSD is not necessary for doing universal quantum computation using 3D surface codes. In addition, we find that the error threshold of 3D surface codes is comparable to the threshold of 2D surface codes. Finally, we propose fault-tolerant quantum computing architectures based on 3D surface codes. We estimate that the resource requirements of one of our architectures could be competitive with the requirements of 2D surface code architectures (given certain assumptions).

In the remainder of this chapter, we review background material from the research literature that is a prerequisite for understanding the results we have just outlined. Firstly, we give a brief overview of a formalism for describing quantum error processes. Secondly, we discuss how to protect quantum information from such errors using quantum error-correcting codes. Thirdly, we consider the problem of processing encoded quantum information in a way that is resilient to errors. Finally, we conclude with a discussion of two leading quantum computing architectures.

1.1 Quantum information and quantum errors

We consider quantum information stored using qubits. A qubit is a two-level quantum system which we model as a normalized vector in the two-dimensional complex Hilbert space $\mathbb{C}^2$. We can write the state of an arbitrary qubit $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$, where $\{|0\rangle = (1,0)^T, |1\rangle = (0,1)^T\}$ are called computational basis states and $|\alpha|^2 + |\beta|^2 = 1$, where $\alpha, \beta \in \mathbb{C}$. We describe the states of multiple qubits using tensor products of single qubits. We use density operators to describe qubits whose state we only partially know. Suppose that we know that a qubit is in one of a number of states $|\psi_j\rangle$, each with probability p_j. We describe this situation using the density operator $\rho = \sum_j p_j |\psi_j\rangle\langle\psi_j|$. If all the p_j are zero except for one, then the qubit is in a pure state, otherwise it is in a mixed state. To describe multiple qubits, we again combine density operators using the tensor product.

To characterize the errors that occur in quantum computers, we use the quantum channel formalism [16]. Suppose we have some qubits described by a density operator, ρ. A quantum channel, $\mathcal{E}$, maps ρ to $\mathcal{E}(\rho) = \sum_k E_k \rho E_k^\dagger$, where $\sum_k E_k E_k^\dagger = I$. The E_k are called Kraus operators [17, 16]. Unitary evolution is a special case where there is only one Kraus operator. Using the quantum channel formalism, we can also describe more general transformations.

Let us examine some common examples of quantum channels. Consider a single qubit described by the density operator, ρ. The depolarizing channel, $\mathcal{E}_D$, is defined as follows:

$$\mathcal{E}_D(\rho) = (1-p)\rho + \frac{p}{3}(X\rho X + Y\rho Y + Z\rho Z), \tag{1.1}$$

where $p \in [0,1]$ and X, Y and Z are the Pauli matrices:

$$X = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}, \quad Y = \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix}, \quad Z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}. \tag{1.2}$$

The depolarizing channel describes an error model where a randomly-chosen Pauli error is applied to a qubit with probability p. An error model where a bit-flip error (Pauli X) happens with probability p is described by the bit-flip channel,

$$\mathcal{E}_X(\rho) = (1-p)\rho + pX\rho X. \tag{1.3}$$

Similarly, an error model where a phase-flip error (Pauli Z) happens with probability p is described by the phase-flip channel,

$$\mathcal{E}_Z(\rho) = (1-p)\rho + pZ\rho Z. \tag{1.4}$$

When we refer to one of $\mathcal{E}_D$, $\mathcal{E}_X$ or $\mathcal{E}_Z$ acting on many qubits, we mean the channel acting independently on each qubit. We do not expect the three error channels we have defined to be perfectly accurate descriptions of the errors that occur in actual quantum computers. However, at present it is still unclear which quantum system(s) will eventually be used to build a fault-tolerant quantum computer. So, it is useful to be able to compare the performance of different fault-tolerant protocols in a way that is relatively agnostic to the physical realization of the qubits. The error channels we have defined satisfy this requirement, and capture some of the features (e.g. decoherence and relaxation to the ground state) of the noise we observe in real quantum computers.

1.2 Protecting quantum information

To protect the information stored in multiple qubits from errors, we use quantum error-correcting codes, which were introduced by Shor [18] and Steane [19]. There are other techniques that we can use to mitigate errors, e.g. dynamical decoupling [20] or using decoherence-free subspaces [21]. However, these error-mitigation techniques do not allow us to suppress errors arbitrarily, as some error-correcting codes do. Therefore, so we concentrate on such codes in this thesis. We first consider a general class of codes called stabilizer codes, which were first defined by Gottesman [22]. Next, we focus on a family of stabilizer codes called surface codes. Finally, we review the definition of subsystem codes, a class of codes that generalize stabilizer codes.

To begin, we recall the definition of the n-qubit Pauli group, $\mathcal{P}_n$:

$$\mathcal{P}_n = < i, X_j, Y_j, Z_j : j \in \{1, \ldots, n\} > \tag{1.5}$$

where X_j denotes X acting on the qubit indexed by j etc. A stabilizer code is specified by a stabilizer group, $\mathcal{S}$, which is an Abelian subgroup of the n-qubit Pauli group that does not contain the negative identity matrix, i.e. $-I \notin \mathcal{S}$ [22, 23]. We will use $\mathcal{S}$ to refer to the stabilizer group and the stabilizer code interchangeably. The encoded (or logical) states of $\mathcal{S}$ are the states that are stabilized by $\mathcal{S}$, i.e. $\{|\psi\rangle \in \mathbb{C}^{2^n} : S|\psi\rangle = |\psi\rangle \forall S \in \mathcal{S}\}$. Let m be the rank of $\mathcal{S}$, i.e. the cardinality of a minimum generating set of $\mathcal{S}$. Then the stabilizer code specified by $\mathcal{S}$ encodes the state of $k = n - m$ logical qubits in the state of n physical qubits. We use a bar to differentiate between logical qubit states, $|\overline{\psi}\rangle \in \mathbb{C}^{2^k}$, and physical qubit states, $|\psi\rangle \in \mathbb{C}^{2^n}$.

The logical operators of a stabilizer code act on the encoded states of the code. The logical Pauli operators are contained in the normalizer of the stabilizer group in the Pauli group, $N_{\mathcal{P}_n}(\mathcal{S})$, which is the set of Pauli group elements that commute with the stabilizer group. We note that $N_{\mathcal{P}_n}(\mathcal{S})$ itself contains $\mathcal{S}$. These are the operators that act as the (logical) identity on the logical states of the code. Operators that are in $N_{\mathcal{P}_n}(\mathcal{S})$ but not in $\mathcal{S}$ act non-trivially on the logical states of the code. We use a bar to denote logical operators that act on encoded states, e.g. $\overline{X}|\overline{0}\rangle = |\overline{1}\rangle$. The code distance (or distance) of a stabilizer code, d, is equal to the smallest weight non-trivial logical operator. The weight of an operator acting on n qubits is simply the number of qubits the operator acts on non-trivially. We use the shorthand $[[n, k, d]]$ to summarize the properties of a stabilizer code, where n is the number of physical qubits, k is the number of logical qubits, and d is the code distance.

To correct errors using a stabilizer code, we measure a generating set of the stabilizers. Suppose we have an encoded state $|\overline{\psi}\rangle$ and qubit j experiences a bit-flip error. The subsequent state of our code is $X_j|\overline{\psi}\rangle$. Unless X_j is a logical operator of the code (we assume this is not the case), it will necessarily anti-commute with one (or more) of the stabilizer generators. Therefore, when we measure this stabilizer we will get a -1 outcome which gives us some information about the error. In general, the list of stabilizer measurement outcomes is called the (error) syndrome. We call an all-ones syndrome a trivial syndrome. We use a classical algorithm called a decoder to process the syndrome and return a correction operator. For the error correction to be successful, we require that the product of the error and the correction operator is a stabilizer. If this is the case, the system will be returned to the same logical state it was in before the error happened. We note that in quantum codes two different errors can have the same syndrome. Such errors are called degenerate errors and codes with this property are called degenerate codes.

The error correction procedure we described for bit-flip errors also works for general qubit error channels. Consider an arbitrary unitary error acting on qubit j, U_j. The n-qubit Pauli group is a basis for n-qubit unitary matrices, so we can expand $U_j = c_I I + c_X X_j + c_Y Y_j + c_Z Z_j$, where $c_j \in \mathbb{C}$. We assume that the non-trivial Pauli operators are not logical operators. Therefore, when we measure the stabilizer generators after the error U_j has occurred, we will either collapse to the I term in the superposition and observe a trivial syndrome or we will observe a non-trivial syndrome. In the second case, we will have collapsed to a case where a Pauli error occurred. For general qubit error channels, we can make the same argument for each of the Kraus operators. We note that the above argument does not hold for error channels where qubits can leak out of the computational subspace. Errors can still be corrected in this case, although the standard stabilizer error correction procedure we described above will need to be modified [22, 24].

In this thesis, we exclusively study a further subclass of stabilizer codes called Calderbank-Shor-Steane (CSS) codes [23, 25], which are named after their inventors. In a CSS code, we can split the stabilizer group into two subgroups, $\mathcal{S}_X$ and $\mathcal{S}_Z$. The subgroup $\mathcal{S}_X$ ($\mathcal{S}_Z$) contains operators that are tensor products of X (Z) and I. The union of a generating set of $\mathcal{S}_X$ and a generating set of $\mathcal{S}_Z$ generates the entire stabilizer group. This structure means that, in a CSS code, we can correct X-errors and Z-errors independently. To correct X (Z) errors, we measure a generating set of $\mathcal{S}_Z$ ($\mathcal{S}_X$) and process the syndrome using a decoder to obtain a correction. This is useful because correcting either X-errors or Z-errors on their own is essentially a classical error correction problem, so we can easily adapt existing decoders for classical codes to the quantum setting. We note that since $Y = iXZ$, to correct any Pauli error we only need to correct X-errors and Z-errors.

1.2.1 Surface codes

So far, we have discussed quantum codes in a relatively abstract way. In this section, we describe a class of topological codes called surface codes, which were introduced by Kitaev [8]. These codes can be understood as toy models of topological order, where a quantum state is said to be topologically ordered if it cannot be prepared by applying a constant depth circuit to a product state [26]. We begin by reviewing the definition of the 2D surface code [8, 9, 10], before discussing its generalization to higher dimensions.

Consider a planar 2D lattice in Euclidean space. We place a single qubit on each of the edges of the lattice. These are the physical qubits of the 2D surface code. We associate X-stabilizer generators with the vertices of the lattice, i.e. for each vertex v, we have the following stabilizer generator:

$$A_v = \bigotimes_{e: v \in e} X_e, \tag{1.6}$$

where X_e denotes an X-operator acting on the qubit on edge e. Similarly, for each

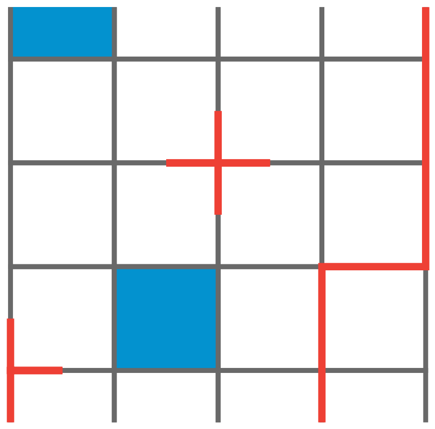


Figure 1.1: The stabilizers and boundaries of a 2D surface code. We highlight two Z-stabilizers (blue squares) and X-stabilizers (red crosses). The top/bottom boundaries are rough boundaries and the left/right boundaries are smooth boundaries. The Z (X) stabilizers have reduced weight on the rough (smooth) boundaries. We also show a logical Z-operator (red string from one rough boundary to the other).

face f, we have the following Z-stabilizer generator:

$$B_f = \bigotimes_{e \in f} Z_e. \tag{1.7}$$

Given these definitions, we see that surface codes are an example of CSS codes. The number of logical qubits encoded in a 2D surface code depends on the boundaries of the lattice. We consider lattices with two types of boundaries: rough boundaries and smooth boundaries (see Figure 1.1). Logical Z (X) operators are strings of physical Z (X) operators that stretch from one rough (smooth) boundary to another. The structure of the logical operators tells us why these codes are called topological codes — the only way to access the information stored in a 2D surface code is to measure a subset of physical qubits along a topologically non-trivial path from one boundary to another. We consider 2D surface codes with two rough boundaries and two smooth boundaries where the two types of boundary alternate. Such codes have one logical qubit and their code distance is equal to the shortest path between boundaries of the same type.

Suppose we have a 2D surface code defined on some lattice $\mathcal{L}$. It is often useful to consider an equivalent 2D surface code defined on the dual lattice, $\mathcal{L}^*$. Given a 2D lattice without boundaries (e.g. a cellulation of the torus), we can construct its dual lattice using the following procedure. First, we create a vertex at the centre

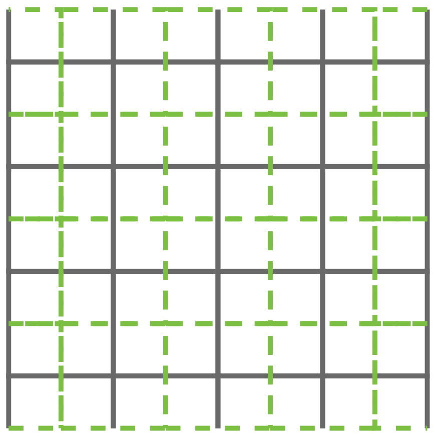


Figure 1.2: The primal lattice (solid grey) and dual lattice (dashed green) of a 2D surface code.

of each of the faces of $\mathcal{L}$. Then we connect these new vertices with an edge if the corresponding faces of $\mathcal{L}$ share an edge. We can also define a dual lattice for 2D lattices with boundaries, although this is slightly more involved [27]. In the dual lattice, qubits are still on edges, but X-stabilizers are associated with faces and Z-errors are associated with vertices. Rough boundaries and smooth boundaries are also interchanged when moving to the dual lattice. Figure 1.2 shows an example of the same 2D surface code in both the primal (original) lattice and dual lattice pictures.

We now discuss 2D surface code decoding. Analysing this problem is easier if we make an analogy to condensed matter physics, as was first noticed by Dennis et al. [28]. One can define a Hamiltonian that corresponds to a given surface code:

$$H = -\sum_{v} A_v - \sum_{f} B_f. \tag{1.8}$$

The (topologically degenerate) ground states of this Hamiltonian are the encoded states of the surface code and the excited states are those with unsatisfied stabilizers (stabilizers with -1 eigenvalues). Therefore, in this picture, we can interpret unsatisfied stabilizers as excitations. Usually, unsatisfied X-stabilizers are called electric charges and unsatisfied Z-stabilizers are called magnetic fluxes [28]. This is because the phase that is acquired (by the wave function) when transporting one excitation around an excitation of the other type is analogous to the phase that is acquired when an electric charge is transported around a magnetic flux in the

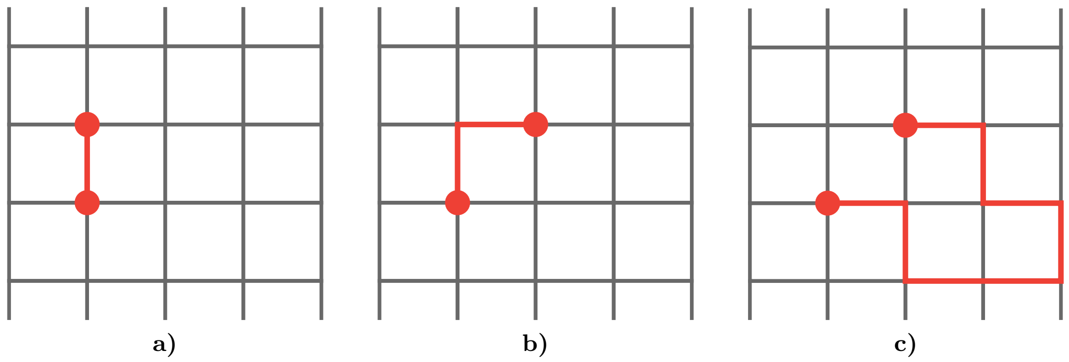


Figure 1.3: Errors in 2D surface codes. In Figure 1.3a, we show the syndrome (red circles) of a single Z-error (red edge). And in Figure 1.3b, we show the syndrome of a larger Z-error. Surface codes are degenerate codes, so different errors can have the same syndrome. In Figure 1.3c, we show a higher weight error with the same syndrome as the error in Figure 1.3b.

Aharonov-Bohm effect [29, 30, 28]. A single Z-error in the bulk of the lattice creates a pair of electric charges (see Figure 1.3). We can also create single electric charges at the rough boundaries (we note that this is sometimes used to define the rough boundaries).

Suppose that we apply the phase-flip channel independently to all the physical qubits in a 2D surface code. We can interpret the syndrome of such an error as a collection of electric charges. A valid correction for such a syndrome is an operator that removes all the electric charges from the system, either by annihilating pairs or by moving single charges to a rough boundary. The standard decoder used for finding such a correction in the minimum-weight perfect-matching algorithm (MWPM) [28, 31, 32], which returns a correction operator with minimum weight. The X-error decoding problem is completely analogous because X-errors and syndromes in the dual lattice have the same structure as Z-errors and syndromes in the primal lattice.

To quantify the performance of a family of quantum codes, we use the concept of an error threshold. Suppose we have a family of codes with increasing size, e.g. surface codes defined on $L \times L$ lattices. Given a particular noise model parameterized by an error probability, p, and a decoder, the error threshold of a code family, p_{th}, is the value of p below which increasing the size of the code exponentially reduces the probability of a logical error (after decoding). If we make some reasonably strong assumptions about the noise, it is sometimes possible to prove that a code family has an error threshold, for example see [33, 34]. When this is not possible, we can

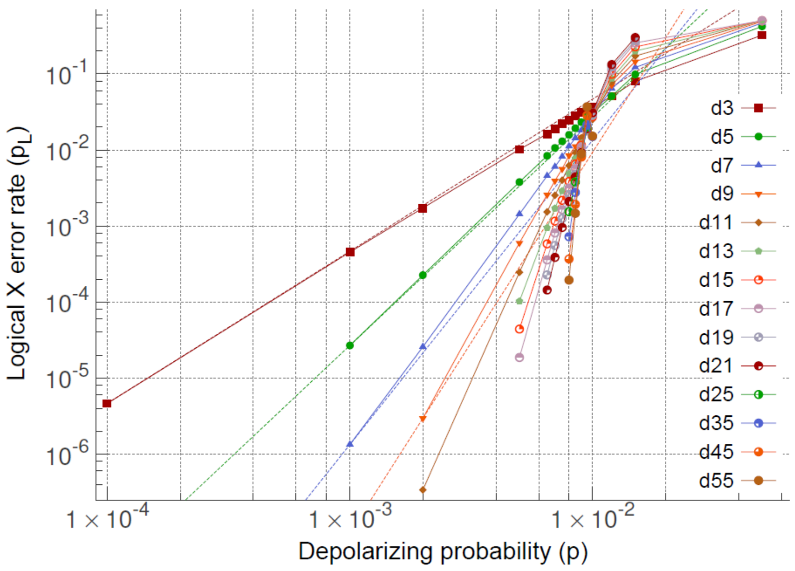


Figure 1.4: A plot showing the error threshold of the 2D surface code. The logical error rate, p_L, is plotted as a function of the depolarizing error probability, p, for different code distances. The error threshold, $p_{th} \approx 1\%$, is the value of p where the different curves intersect. Figure adapted from Fowler et al. [35]

use Monte Carlo simulations to estimate the logical error probability as a function of the physical error probability for various code sizes. The point where the data for different sizes intersect gives us an estimate of the error threshold (see Figure 1.4 for an example).

In the error models we have considered so far, we have assumed that stabilizer measurements are perfect, but this is unlikely to be the case in actual quantum computers. Therefore, we also want to find the error threshold of a code family when stabilizer measurements are unreliable. Error models of this type are often called phenomenological error models. In 2D surface codes, the standard approach to decoding in the presence of measurement errors is to repeat the stabilizer generator measurements for a number of times that scales with the code distance. We then use a decoder to process the syndrome history (the record of all the stabilizer measurements) and return a correction. Using this approach, decoding a 2D surface code in the presence of measurement errors is equivalent to decoding errors in a 3D graph (where one dimension represents time) rather than a 2D graph, as was pointed out by Dennis et al. [28]. We can also use the MWPM algorithm to decode in this case. In addition, we can consider more detailed noise models that take into account the circuits used for measuring the stabilizer generators. Estimates of the error threshold of the 2D surface code for noise models with unreliable measurements range from 0.5% to 1.4% depending on the details of the error model (see [14] and references therein).

Given that the code distance of an $L \times L$ surface code is equal to $L = O(\sqrt{n})$ (where n is the number of physical qubits in the code), it is somewhat surprising that this code family has an error threshold. Suppose that a bit-flip error affects each physical qubit independently with probability p. Then, as we increase L, the expected number of errors $np \approx L^2 p$ will become larger than L, which is equal to the code distance. Therefore, we may expect that the probability of a logical error occurring increases as L increases. However, this does not happen as the proportion of weight L errors that cause a logical error becomes smaller as L increases [36].

The 2D surface code can be readily generalized to higher dimensions, as was first noted by Dennis et al. [28]. In later chapters, we study the properties of 3D surface codes in detail. Here, we review the definition of 3D surface codes and contrast them with 2D surface codes. In addition, we give a brief overview of prior work on these codes.

Consider a 3D lattice with boundaries in Euclidean space. We place qubits on the edges of the lattice, we associate X-stabilizers, A_v, with the vertices and we associate Z-stabilizers, B_f, with the faces. As in the 2D case, it is useful to define a Hamiltonian which corresponds to the code, $H = -\sum_v A_v - \sum_f B_f$. This Hamiltonian allows us to interpret unsatisfied stabilizers as excitations. However, unlike the 2D case, the X-stabilizer excitations (electric charges) are different from the Z-stabilizer excitations (magnetic fluxes). Specifically, the electric charges are associated with 0D lattice elements (vertices in the primal lattice) whereas the magnetic fluxes are associated with 1D lattice elements (edges in the dual lattice). This difference has consequences for the error correction properties of the code, as we show in Chapter 3. In the bulk of the lattice, we can only create pairs of electric charges (vertices connected by edges) and closed loops of magnetic flux (edges that form the boundary of a set of faces). However, we can create single electric charges on the rough boundaries, and open loops of magnetic flux can terminate on the smooth boundaries (see Figure 1.5). For now, we use these properties to define the rough and smooth boundaries. In later chapters, when we consider specific lattices,

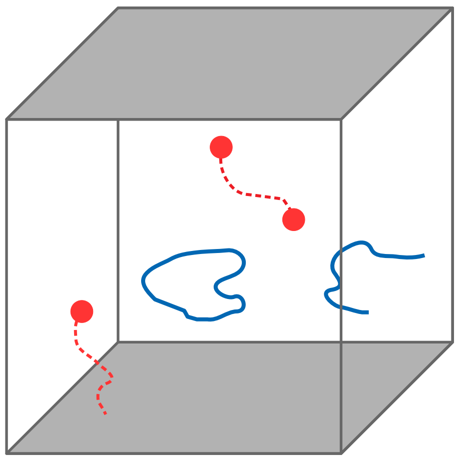


Figure 1.5: Errors and syndromes in 3D surface codes. The rough boundaries are shaded and the smooth boundaries are clear. We show examples of Z-errors (dashed red lines) and their syndromes (red circles). We also show the syndromes of X-errors (solid blue lines). The errors causing these syndromes are collections of faces in the dual lattice whose boundary is equal to the blue lines.

we use a more explicit definition of the boundaries.

We restrict our attention to lattices that are tessellations of the cube with four smooth boundaries and two rough boundaries, where the rough boundaries are on opposite sides of the cube (e.g. the lattice shown in Figure 1.5). Surface codes defined on such lattices encode a single logical qubit. The logical operators of such a code can be understood in terms of electric and magnetic excitations. We can implement a logical Z-operator by creating a pair of electric charges in the bulk and annihilating them on opposite rough boundaries. Therefore, logical Z-operators are strings of (physical) Z-operators that terminate on opposite rough boundaries. Similarly, we can implement a logical X-operator by creating a loop of magnetic flux in the bulk of the lattice and expanding the loop until it annihilates on all four smooth boundaries. This implies that logical X-operators are membranes of (physical) X-operators with a boundary that is a cycle around the four smooth boundaries. Finally, we note that the code distance of the 3D surface codes we consider is equal to the length of the shortest path between the two rough boundaries.

The different structure of the X-excitations and Z-excitations in 3D surface codes means that decoding X-errors and Z-errors are different problems. Decoding Z-errors is the same as decoding errors in 2D surface code, except the lattice is now

3D. The X-error decoding problem is easier to state if we switch to the dual lattice. In this (equivalent) picture, qubits are on faces, X-stabilizers are associated with cells and Z-stabilizers are associated with edges. Therefore, the syndromes of X-errors are loops of edges (which can be open or closed). Given a syndrome, finding a correction is equivalent to finding a collection of faces whose boundary equals the syndrome. We return to the topic of decoding 3D surface codes in Chapter 3.

Like the 2D case, 3D surface codes are toy models of topological order. In fact, 3D surface codes were originally studied in this context [37, 38]. Prior work on 3D surface codes in the context of quantum computation has focussed on their relationship to other topological quantum codes [39, 40]. There has also been work on decoding 3D surface codes [41, 42, 43, 44] and on understanding their non-Pauli logical operators [39, 45]. We give a fuller account of this work in later sections and chapters.

In more than three dimensions, there are multiple possible definitions of surface codes. In 4D, one possible definition is to place qubits on the edges of a 4D lattice, associate X-stabilizer generators with vertices and associate Z-stabilizer generators with faces. However, there is an inequivalent definition where we place qubits on faces, associate X-stabilizer generators with edges and associate Z-stabilizer generators with cells. The second definition is more interesting, because a 4D surface code with qubits on faces is an example of a passive quantum memory [28, 46]. Such a memory is the quantum analogue of a hard disk, i.e. a quantum memory that is stable against thermal fluctuations and therefore does not require active error correction to preserve information. It is still an open question as to whether such a passive quantum memory exists in 3D [47].

In D dimensions, we can define a surface code with qubits on i-cells, X-stabilizers on $(i-1)$-cells and Z-stabilizers on $(i+1)$-cells, where a i-cell is a i-dimensional cell. For example, 0-cells are vertices, 1-cells are edges, 2-cells are faces etc. As long as $i \in \{1, \ldots, D-1\}$, then we will have a valid surface code. We refer to a surface code with qubits on i-cells as a type-i surface code. In the dual lattice of a type-i surface code, qubits will be associated with $(D-i)$-cells, X-stabilizers will be associated with $(D-i+1)$-cells and Z-stabilizers will be associated with $(D-i-1)$-cells. We recall that as surface codes are CSS codes, we can correct X- and Z-errors independently. For a given type-i surface code, we say that X (Z) errors have m-dimensional syndromes, where m is the smallest dimensional lattice element (m-cell) to which Z (X) stabilizers are associated in either the primal or dual lattice. That is, $m = \min(i-1, D-i+1)$ for Z-errors and $m = \min(i+1, D-i-1)$ for X-errors. For example, consider type-2 4D surface codes. In the primal lattice, qubits are on faces (2-cells), X-stabilizers are associated with edges (1-cells), and Z-stabilizers are associated with cells (3-cells). And in the dual lattice, qubits are still on faces, but the support of the stabilizer types is exchanged. Therefore, both X- and Z-errors in type-2 4D surface codes have 1-dimensional syndromes.

1.2.2 Subsystem codes

Subsystem codes, introduced by Poulin, are a generalization of stabilizer codes to non-commuting stabilizer groups [48]. In a subsystem code, the encoded qubits separate into two sets: the logical qubits and the gauge qubits. We only store information in the logical qubits, but the gauge qubits give the code additional structure which can be useful for error correction. A subsystem code with n physical qubits is defined by its gauge group $\mathcal{G}$, a (possibly non-Abelian) subgroup of the n-qubit Pauli group. The stabilizer of a subsystem code, $\mathcal{S} = \mathcal{Z}(\mathcal{G})$, is the centre of the gauge group, i.e. the elements of the gauge group that commute with everything in the gauge group. We note that a subsystem code with an Abelian gauge group is a stabilizer code. In a subsystem code, there are two types of logical operators: bare logical operators and dressed logical operators. Bare logical operators are the elements of the Pauli group that commute with the gauge group but are not in the gauge group. These logical operators act on the logical qubits and don't have an effect on the gauge qubits. Dressed logical operators are the elements of the Pauli group that commute with the stabilizer but are not in the gauge group. These operators act on the logical qubits and the gauge qubits.

Error correction in subsystem codes works in a slightly different way to stabilizer codes. Instead of measuring the stabilizers, we measure a generating set of gauge operators. As stabilizers are products of gauge operators, we can infer the values of the stabilizers from the gauge operator measurements. For some subsystem codes, the same stabilizer operator is the product of different sets of gauge operators. Therefore, the measurement outcomes of the gauge operators redundantly tell us the values of the stabilizers. Consequently, we can avoid having to do multiple rounds of measurement to get reliable estimates of the eigenvalues of the stabilizer generators. This is an example of a property of quantum codes called single-shot error correction [49, 50, 51]. A code with single-shot error correction is simply a code where we can obtain a reliable error syndrome using $O(1)$ stabilizer (or gauge operator) measurements.

In this section, we have given a brief overview of quantum error-correcting codes, with a particular focus on surface codes, a family of topological quantum codes. Codes from this family are some of the most promising candidates for protecting quantum information from decoherence caused by interactions with the environment. This is chiefly due to their high error threshold, robustness to disorder and relatively simple structure. However, a family of codes that is good at protecting quantum information will not necessarily be well-suited to processing encoded quantum information. In fact, in the next section we explain how these two requirements can come into conflict.

1.3 Processing encoded quantum information

If we want to build a fault-tolerant quantum computer, we need find methods for processing quantum information in a way that is resilient to errors. Shor was the first to tackle this problem [52], proposing fault-tolerant protocols based on error-correcting codes. There are a range of possible definitions of fault-tolerance, but the one we use here is as follows. We call a protocol fault-tolerant if one failure during the protocol leads to at most one physical qubit error in each code block (a single $[[n, k, d]]$ code), where a failure can be a single-qubit error, a gate error, a measurement error etc. This captures the idea of errors not spreading too much during the protocol. We begin this section by giving an overview of the requirements of a fault-tolerant quantum computing architecture, using the 2D surface code as an example. We then focus on schemes for doing fault-tolerant logical gates in topological codes, giving an overview of existing results in the research literature. Finally, we consider a quantum computing architecture based on a class of topological codes called colour codes.

1.3.1 Fault-tolerance with 2D surface codes

There are some basic operations which we need to be able to implement fault-tolerantly if we want to execute an arbitrary quantum algorithm with imperfect hardware. These operations are: preparation of encoded $|0\rangle$ states, measurement in the Z-basis, implementation of a universal gate set and error correction. First, we consider fault-tolerant error correction. In Section 1.2.1, we discussed decoding 2D surface codes when measurements are unreliable. However, we also need to ensure that measuring stabilizer generators does not corrupt the encoded data in an uncontrolled way.

Consider a 2D surface code X-stabilizer, $X^{\otimes 4}$. We can measure this stabilizer using the circuit shown in Figure 1.6. If an X-error happens to the ancilla qubit

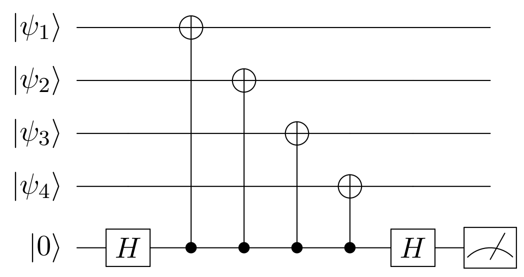


Figure 1.6: A circuit for measuring an $X^{\otimes 4}$ stabilizer. $|\psi_i\rangle$ are the physical qubits of the code, $H = \frac{1}{\sqrt{2}}(X + Z)$ is the Hadamard gate, and the ancilla is measured in the Z-basis.

at the start of the circuit, then each of the physical qubits experiences an X-error due to the controlled-NOT ($CNOT$) gates. In general, when using the method in Figure 1.6 to measure stabilizers, one ancilla error can cause up to w physical qubit errors for a weight w stabilizer. We can prevent this from happening using a variety of fault-tolerant gadgets, which allow us to replace naive stabilizer measurement circuits with fault-tolerant circuits (e.g. [52, 53, 54, 55, 56, 57]). These circuits have the same effect as the original circuits, but they limit the propagation of errors. Interestingly, using fault-tolerant stabilizer measurement circuits is not necessary in 2D surface codes [28]. An intuitive explanation for this is that although single ancilla errors can cause weight two physical qubit errors (weight-four errors are stabilizers and weight-three errors are equivalent to weight-one errors up to a stabilizer), these errors will be on nearest-neighbour qubits. We do not expect such errors to lead to logical errors due to the topological protection given by the code. However, we do expect these correlated errors to degrade the error threshold, which is what has been seen in simulations [14].

We now consider the task of fault-tolerantly preparing encoded computational basis states. Dennis et al. proposed a method for preparing encoded $|0\rangle$ (and $|+\rangle$ states) in 2D surface codes [28], which we review here. To begin with, we assume that state preparation and measurement of physical qubits are both perfect. First, we prepare all the physical qubits of the code in the $|0\rangle$ state. This state is already a $+1$ eigenstate of all the Z-stabilizers. Next, we measure the X-stabilizer generators, each of which will have a random outcome. Finally, we process the syndrome and apply a correction. This correction may well result in the application of a $\overline{Z}$-operator, but this has no effect on the encoded state because the code started off in a $+1$ eigenstate of $\overline{Z}$ and measuring X-stabilizers will not change this eigenvalue. If state preparation and measurement are imperfect, then we have to modify the procedure somewhat. We still attempt to prepare every physical qubit in the $|0\rangle$ state, but instead of just measuring X-stabilizers we measure all the stabilizer generators d times, where d is the code distance. The final step of the procedure is to use an efficient decoder (e.g. MWPM for 2D surface codes) to compute a correction. We again might apply a $\overline{Z}$-operator, but the probability of applying a $\overline{X}$-operator will be exponentially suppressed in the size of code, as long as we are below threshold. We can prepare a logical $|+\rangle$ state using an analogous procedure where we start with all the physical qubits in the $|+\rangle$ state.

The penultimate requirement we consider is being able to measure encoded qubits in the Z-basis fault-tolerantly. Again, there is a simple procedure for achieving this with surface codes [28]. We assume that we have a code with a single encoded qubit for simplicity. To measure this qubit in the Z-basis we first measure all the physical qubits in the Z-basis. Next, we calculate the values of all the Z-stabilizers using the outcomes of the physical qubit measurements. The final step is to use an efficient decoder to process the stabilizer measurements and apply a correction (which in this context just means flipping some physical qubit measurement outcomes). We then compute the eigenvalue of any $\overline{Z}$-operator. This procedure works because single measurement errors do not lead to unsatisfied stabilizers that are very far from each other in the lattice. So unless the measurement errors are correlated over a distance comparable to the lattice size, we can deal with them in the same way that we deal with physical qubit errors before the measurements. To measure in the X-basis, we repeat the above procedure except with X instead of Z.

The final demand we make of a quantum computing architecture is that it should be possible to implement a universal set of quantum gates fault-tolerantly. A discrete set of gates is called universal if we can approximate any unitary operation using gates from the set. Importantly, the Solovay-Kitaev theorem guarantees that this approximation is efficient [58, 59]. Let us consider some examples of universal gate sets. The most common universal gate set is called Clifford+T. Clifford gates are unitary operators that map Pauli operators to Pauli operators. The two-qubit Clifford group can be generated by $CNOT = I_2 \oplus X$, the Hadamard gate $H = (X + Z)/\sqrt{2}$, and the phase gate $S = \text{diag}(1, i)$, where I_j is the $j \times j$ identity matrix. We call any (non-Pauli) gate that is not in the Clifford group a non-Clifford gate. The T-gate is a non-Clifford gate with the following form: $T = \text{diag}(1, \exp(i\pi/4))$. Adding any non-Clifford gate to the Clifford group gives us a universal set of gates [60]. We can also construct universal sets without using the Clifford group, e.g. the Hadamard gate and the controlled-controlled-Z gate ($CCZ = I_6 \oplus Z$) together are universal [61].

Given a universal gate set, we need to find a fault-tolerant implementation of each gate in the set. One natural class of fault-tolerant gates is transversal gates. Suppose we want to implement a gate on j logical qubits and we use m $[[n,k,d]]$ codes to protect the logical qubits, where $m = \lceil j/k \rceil$. We call each of the individual codes a code block. Then, a transversal gate (or operation) is a depth-one circuit that does not entangle qubits in the same code block. Examples of transversal gates include tensor products of single-qubit unitary operators (e.g. logical Pauli gates in stabilizer codes).

Unfortunately, the power of transversal gates is limited, as shown by the Eastin-Knill theorem:

Theorem 1 ([62]) *Suppose that an error-correcting code can detect an error on an arbitrary physical qubit. Then its set of transversal encoded gates is not universal.*

We give an idea of the physical intuition behind this result below, following the argument given in [63]. Assume that some code has a universal set of transversal gates. Then, we can implement a continuous group of rotations using a subset of the transversal gates. These gates must conserve the charge J that is associated with the continuous rotation group. For a given code word, there will be a conserved charge in the logical qubit space, $\overline{J}$, and a corresponding conserved charge in the physical qubit space, $J = \sum_i J_i$, which is a sum of single-qubit terms due to the transversal nature of the gates. The environment has access to the local reduced states and can therefore estimate the value of the conserved charge and thereby access the logical information stored in the code. This fact implies that the code cannot correct single-qubit erasures, which is close in spirit to being able to detect single-qubit errors.

The Eastin-Knill theorem exposes a deep conflict between protecting quantum information and processing quantum information. However, there are methods for avoiding the constraints of this theorem. Firstly, one can fault-tolerantly switch between different codes with different sets of transversal gates. Each code individually will not have a universal and transversal set of gates, but the union of the transversal gates of both codes can be universal. Alternatively, the Eastin-Knill theorem can be circumvented by using measurements and classical feed-forward, as is the case in magic state distillation schemes. Finally, we note that the theorem does not apply to non-transversal but still fault-tolerant gates (e.g. non-transversal constant depth unitaries in topological codes).

To begin our discussion of fault-tolerantly implementing a universal gate set with 2D surface codes, we review two ways of encoding a large number of qubits in a single layer of surface code. The first encoding method is defect-based encoding, which was introduced by Raussendorf et al. [64]. Suppose we have a large layer of surface code with only smooth boundaries. Such a surface code has no encoded qubits. We can create defects in the bulk of the layer by ceasing to measure some of the stabilizers. This introduces additional degrees of freedom into the system, which we can use to encode logical qubits [11, 15]. We can also encode qubits in twist defects [65, 66] (a twist in this context is the end-point of a lattice dislocation). Using twist defects, it is possible to implement the Clifford group fault-tolerantly in 2D surface codes [66].

The second method for encoding a large number of qubits in a single sheet of surface code is called lattice surgery. This encoding method was introduced by

Horsman et al. [67]. In a lattice-surgery architecture, we split a large sheet of surface code into multiple smaller patches of surface code, each encoding a single qubit. We can merge and split different patches by changing the stabilizers that we measure between patches [67]. Using a lattice-surgery encoding, we can also realize the Clifford group fault-tolerantly [68]. Out of the two encodings, lattice surgery is currently favoured because it requires fewer physical qubits to encode a logical qubit with the same code distance [69, 70].

We have discussed two methods for implementing the Clifford group in a 2D surface code architecture. However, to achieve universality, we need a method for implementing a non-Clifford gate. We do not need to worry about the Eastin-Knill theorem, because the implementations of the Clifford group we have described do not rely exclusively on transversal gates. However, the 2D surface code does not have a transversal non-Clifford gate. Almost all codes with such gates have a property called tri-orthogonality, a concept introduced by Bravyi and Haah:

Definition 1 (Tri-orthogonal code [71])**.** Let the H be a $m \times n$ binary matrix describing the support of the X-stabilizer generators of a $[[n, k, d]]$ CSS code. That is, each row in H represents a stabilizer generator, and if $H_{ij} = 1$ then the generator corresponding to row i acts non-trivially on qubit j. Otherwise, $H_{ij} = 0$. In addition, let L be a $k \times n$ binary matrix describing the support of the logical X-operators of the code. The code is called tri-orthogonal if, and only if, $G = \left(\frac{H}{L}\right)$ has the following properties:

$$\sum_{j=1}^{n} G_{aj} G_{bj} = 0 \mod 2, \tag{1.9}$$

for all pairs of rows $1 \leq a \leq b \leq m$, and

$$\sum_{j=1}^{n} G_{aj} G_{bj} G_{cj} = 0 \mod 2, \tag{1.10}$$

for all triples of rows $1 \leq a \leq b \leq c \leq m$. In other words, the overlap of the supports of any pair of rows in G is even and the overlap of the supports of any triple of rows in G is also even.

From the definition, it may not be immediately clear how tri-orthogonality relates to transversal non-Clifford gates. However, it transpires that that the phases of certain Z-rotations applied to the physical qubits of a tri-orthogonal code sum in

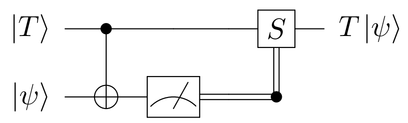


Figure 1.7: A circuit that consumes one $|T\rangle$ state to apply the (non-Clifford) T-gate to a second qubit, $|\psi\rangle$. This circuit is an example of a gate teleportation circuit, a generalization of quantum (state) teleportation [77, 78].

just the right way to implement a logical non-Clifford gate [71, 72]. We will see an example of this in Section 2.2.

Since the 2D surface code is not tri-orthogonal, we must find another way of implementing a non-Clifford gate fault-tolerantly. The standard solution to this problem is known as magic state distillation (MSD), and was introduced by Bravyi and Kitaev [73]. In an MSD protocol, we start with n_i low-fidelity magic states, which we distil into n_f higher-fidelity magic states (where $n_f < n_i$). The most common magic state is the T-state, $|T\rangle = T|+\rangle$. Once we have a high-fidelity encoded T-state, we use a state-injection circuit (e.g. Figure 1.7) to apply the (non-Clifford) T-gate to another encoded qubit, consuming the T-state in the process. The distillation procedure achieves exponential error suppression, so we only need to repeat it for a number of times that is logarithmic in the target infidelity of the magic state(s). In addition, the distillation procedure and the state-injection circuit only use Clifford gates. Therefore, we can implement a MSD protocol using the logical qubits of a quantum code, as long as the code has a fault-tolerant implementation of the Clifford group. For example, MSD can be used to implement a T-gate fault-tolerantly in a 2D surface code architecture [15]. Interestingly, many MSD protocols are based on tri-orthogonal codes [71, 72, 74, 75, 76].

In this section, we have described the protocols that need be realized fault-tolerantly in a quantum computer that is resilient to error. Methods for fault-tolerant state preparation, measurement, error correction and logic can be combined to prove threshold theorems [79, 80, 81]. Suppose that qubit errors, state preparation errors, measurement errors and gate errors occur according to probabilities that are smaller than some threshold p_{th}. Then, the threshold theorem tells us that a fault-tolerant version of any circuit can be constructed, where the overhead of the fault-tolerant circuit is poly-logarithmic in the number of qubits in the original circuit and in the number of gates in the original circuit. This result has been generalized to a wide variety of noise models [82, 83, 84, 85], which gives us confidence that building a fault-tolerant quantum computer is possible in principle.

The threshold theorem tells us that the asymptotic resource cost of fault-tolerance is not too large, but the resource requirements of a fault-tolerant architecture can still be substantial in practice. For example, consider the 2D surface code. We can estimate the resource requirements of a 2D surface code architecture by using a space-time metric. We say that an architecture that requires n physical qubits per logical qubit and d rounds of stabilizer measurement per operation has a space-time overhead of nd (we neglect the cost of efficient classical computation). In a 2D surface code architecture, we require $\approx d^2$ physical qubits per logical qubit, where d is the code distance, which is a function of the physical-qubit error rate and the target logical error rate. Per operation, we need to do $O(d)$ rounds of stabilizer measurement, so the space-time overhead of using 2D surface codes is $O(d^3)$. This figure gives tells us how the overhead scales, but the Big O notation hides some important constants. For example, using 2D surface codes, it has been estimated that the resource cost of implementing a T-gate using MSD is $\sim$ 150–300 times more than the cost of implementing Clifford gates [86]. Reducing this cost is an active field of research [74, 75, 76, 87], though no full-scale analysis has been done for the most recent MSD schemes. In addition, the overhead of MSD has motivated research into fault-tolerant implementations of non-Clifford gates in topological codes that do not require MSD. In the following sections, we review some of the results of these investigations.

1.3.2 Restrictions on non-Clifford gates in topological codes

There is an important theorem about fault-tolerant logical gates in topological codes, which was proved by Bravyi and König [88]. Much of the following discussion follows their article closely. Before stating the theorem, we need to define a generalization of the Clifford group called the Clifford hierarchy [77]. The D'th level of the Clifford hierarchy, $\mathcal{C}_D$, is the set of all unitary operators that map Pauli operators to operators in the $(D-1)$'th level of the hierarchy (under conjugation). That is,

$$\mathcal{C}_D = \{U : U\mathcal{P}_n U^\dagger \subseteq \mathcal{C}_{D-1}\}, \tag{1.11}$$

where $\mathcal{C}_1 := \mathcal{P}_n$ (the Pauli group). From the definition, we see that $\mathcal{C}_2$ is the Clifford group. $\mathcal{C}_3$ contains the T-gate and CCZ gate. We are now ready to state Bravyi and König's theorem:

Theorem 2 ([88]) *Suppose a unitary operator U implementable by a geometrically-local constant-depth quantum circuit preserves the codespace of a topological code on a D-dimensional lattice, $D \geq 2$. Then U implements a gate from the set $\mathcal{C}_D$.*

Constant-depth circuits are a natural class of fault-tolerant gates for topological codes. This is because a constant-depth circuit can only spread an existing error in a region R to a new region of size $R + O(1)$. Also, any error that happens during the constant-depth circuit can cause errors on at most $O(1)$ physical qubits. As topological codes are robust against errors in local regions, if an error is correctable before a constant-depth circuit, it will be correctable afterwards. An implementation of a unitary using a constant-depth circuit is sometimes called a logical-preserving logical operator (LPLO) [45].

One important consequence of Theorem 2 is that the 2D surface code has no non-Clifford LPLOs. We now sketch the proof of the theorem for this special case. Consider a 2D surface code with two rough and two smooth boundaries, encoding a single logical qubit. For any two logical Pauli operators $\overline{P}$ and $\overline{Q}$, we can always find equivalent (up to stabilizers) logical operators that overlap on at most two physical qubits. Now, assume we apply a constant-depth unitary U that preserves the codespace. Let us consider the group commutator of P and Q conjugated by U: $K[P, UQU^\dagger] = P(UQU^\dagger)P^\dagger(UQU^\dagger)^\dagger$. Because P and Q overlap on a constant number of qubits and U is constant depth, the group commutator only has non-trivial support on $O(1)$ qubits. Any operator that preserves the codespace and has $O(1)$ support must be an implementation of $\pm I$ because of the topological protection offered by the code. One can verify that an operator whose group commutator with Pauli operators is trivial must either be a Pauli operator or $\pm I$. Therefore $UQU^\dagger \in \mathcal{P}_n$ for any $Q \in \mathcal{P}_n$, which means that U is in the Clifford group.

Bravyi and König's result has been generalized to subsystem codes [89] and to constant-depth unitary operators that are geometrically non-local [90]. It is important to note that a D-dimensional topological code is not guaranteed to have an LPLO in $\mathcal{C}_D$. For example, the 4D surface code (with qubits on faces) has no non-Clifford LPLOs, as can be seen from a similar argument to the one made above. Interestingly, the structure of the 4D surface code's logical operators makes the code thermally stable [46], but also prevents it from having non-Clifford LPLO. Therefore, the 4D surface code is another example of the conflict between protecting and processing quantum information. Finally, we note that the LPLOs of various topological codes have recently been classified using an analogy between LPLOs and generalized domain walls [91, 92, 45]. In particular, it was recently shown that the (non-Clifford) CCZ gate is an LPLO in the 3D surface code [45].

1.3.3 Colour codes

In this section, we review a family of topological quantum codes that saturate Theorem 2 in all spatial dimensions: colour codes. This code family was introduced by Bombín and Martin-Delgado [93, 94, 95]. To begin, we revise the definition of 2D colour codes, before considering 3D colour codes in more detail. Then, we examine a subsystem code generalization of the 3D colour code. We conclude this section by estimating the space-time overhead of a colour code quantum computing architecture.

In 2D, a colour code can be defined on any three-valent lattice that has three-colourable faces, i.e. one can assign one of three colours to each face of the lattice so that faces sharing an edge have different colours. Figure 1.8 shows the three regular 2D colour code lattices. Qubits are placed on the vertices of the lattice and X-stabilizer generators and Z-stabilizer generators are associated with the faces of the lattice. That is, for each face f, there is an X-stabilizer $A_f = \bigotimes_{v\in f} X_v$ and a Z-stabilizer $B_f = \bigotimes_{v\in f} Z_v$, where $Z(v)$ $(X(v))$ denotes a Pauli Z (X) acting on the qubit at vertex v. Like 2D surface codes, the logical operators of 2D colour codes are supported on topologically non-trivial paths in the lattice. Every 2D colour code has transversal Clifford operators, but the specific operators depends on the boundaries of the code. For example, H and S are transversal for codes defined on lattices that tessellate the triangle, whereas $SWAP \cdot H \otimes H$ and $CZ = I_2 \oplus Z$ are transversal for codes defined on lattices that tessellate the square.

3D colour codes are defined on four-valent lattices whose cells are four-colourable, i.e. the cells of the lattice can each be assigned one of four colours so that no cells that share a face have the same colour. Qubits are placed on the vertices,

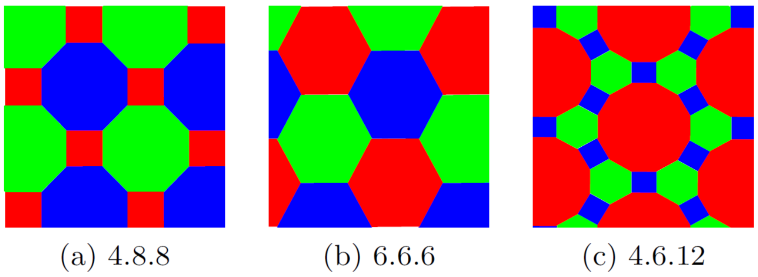


Figure 1.8: The three regular 2D colour code lattices. We use the $(r.s.t)$ shorthand to denote a lattice where an r-gon, an s-gon and a t-gon meet at each vertex. Figure adapted from Landahl et al. [96].

X-stabilizer generators are associated with the cells and Z-stabilizer generators are associated with the faces of the lattice. Unlike their 2D cousins, 3D colour codes do not have a transversal implementation of the Clifford group. However, every 3D colour code is a tri-orthogonal code and consequently has a transversal non-Clifford gate [97, 98, 99]. For example, a 3D colour code defined on a tessellation of the tetrahedron has a transversal T-gate, and a 3D colour code defined on a tessellation of the cube has a transversal CCZ gate.

Given that colour codes have a wider variety of transversal gates when compared with surface codes, it is natural to ask why surface codes are still the leading topological codes. The main reason is that colour codes tend to have lower error thresholds than surface codes. This is because colour codes have higher weight stabilizers than surface codes, which makes errors during the stabilizer measurement procedure more likely and leads to a lower error threshold. Also, decoding colour codes is more complicated than decoding surface codes, because the structure of the excitations in colour codes is more complex [100, 96].

The stabilizer weight problem is more pronounced in higher-dimensional colour codes. The 3D colour code lattice with the lowest average stabilizer weight still has weight twenty-four X-stabilizers. To alleviate this problem, Bombín introduced a subsystem code version of the 3D colour code called the gauge colour code [97]. These codes are defined on 3D colour code lattices and their physical qubits are placed on the vertices. We associate X and Z gauge generators with the faces of the lattice. We recall from Section 1.2.2 that the stabilizer group of a subsystem code is the centre of the gauge group. We can, therefore, deduce that the stabilizer generators of the gauge colour code are associated with the cells of the lattice. In a subsystem code we can find the eigenvalues of the stabilizers by measuring the gauge generators, and in the gauge colour code the gauge operators (faces) are much lower weight than the stabilizers (cells). In addition, the structure of the gauge colour code means that we only need to measure the gauge generators $O(1)$ times to gain a reliable estimate of the eigenvalues of the stabilizers. Therefore, the gauge colour code exhibits single-shot error correction [49].

The gauge colour code also has another useful property: so-called 'dimensional jumps' [101]. A dimensional jump is a fault-tolerant method for switching between a 3D gauge colour code and a 2D colour code defined on its boundary. For example, the gauge colour code could be defined on a lattice that tessellates the tetrahedron and the 2D colour code would be defined on one of the triangular faces of the tetrahedron. As such a 2D colour code has a transversal implementation of the Clifford group and the tetrahedral gauge colour code has a transversal T-gate, the dimensional jump allows us to implement a universal and fault-tolerant set of gates without using MSD. This is an example of circumventing the Eastin-Knill theorem (Theorem 1) by switching between different codes with complementary transversal gates.

The gauge colour code's single-shot error correction property means that we only need to measure the stabilizer generators $O(1)$ times per dimensional jump. Single-shot error correction also enables us to prepare encoded $|0\rangle$ states in constant time. In addition, we can fault-tolerantly measure encoded qubits in the Z-basis in constant time, using an analogous procedure to the 2D surface code measurement procedure (described in Section 1.3.1). Consequently, the number of stabilizer measurements per operation in a gauge colour code architecture is $O(1)$. We need $\approx d^3$ physical qubits per logical qubit in this architecture, so the overall space-time overhead is $O\left(d^3\right)$, which is scales in the same way as the overhead of 2D surface code architectures (see Section 1.3.1).

The error threshold of the gauge colour code has been estimated to be $p_{th} \sim 0.31\%$ [102], which is slightly smaller than the threshold of the 2D surface code. However, we emphasize that the value of the error threshold is not the only figure of merit which should be used when comparing two codes. The quality of the error-suppression below the error threshold is also important. Given that the resource overheads and error thresholds of 2D surface codes architectures and gauge colour code architectures are similar, the answer as to which architecture is preferable will likely depend on the type of qubits used. For example, superconducting qubits fabricated on a 2D chip [103] will be better-suited to a (planar) 2D code but ion-trap qubits may be better-suited to 3D codes due to the availability of non-local gates [104]. We return to this discussion in Chapter 4, where we present an architecture based on 3D surface codes.

The final aspect of colour codes that we discuss is their surprisingly close relationship to surface codes [105, 106, 107, 39, 40, 108]. It transpires that for any D-dimensional colour code, there exists a local Clifford unitary that maps the colour code to D copies of the D-dimensional surface code [39]. This equivalence is useful in a number of ways. For example, the colour code decoding problem can be reduced to the surface code decoding problem [109, 110]. Since surface codes are generally easier to decode than colour codes, the equivalence between the code families gives us better colour code decoders. In addition, the equivalence can be used to implement new logical gates in surface codes [39, 111]. In an article that inspired our work, Kubica et al. showed that it is possible to apply a CCZ gate fault-tolerantly to three 3D surface codes by transforming the surface codes into a 3D colour code with a transversal CCZ [39].

Conclusion

Building a fault-tolerant quantum computer requires us to protect quantum information and process encoded quantum information. In this chapter, we have considered both of these requirements in turn and exposed the tension between them. More specifically, we reviewed the Eastin-Knill theorem, which states that any code that can correct a single error can not have a universal and transversal set of encoded gates. Fortunately, the threshold theorem tells us that even with this tension, building a fault-tolerant quantum computer is possible. We have reviewed two leading quantum computing architecture proposals: one based on 2D surface codes and one based on gauge colour codes. These architectures each have their strengths, and which architecture is better is likely to depend on the physical systems used to realize the components of the computer. However, to make fault-tolerant quantum computers a reality sooner, we need to reduce the resource requirements of the current leading architectures.

In the remainder of this thesis, we build towards proposing a fault-tolerant quantum computing architecture based on 3D surface codes. First, in Chapter 2, we consider processing information encoded in 3D surface codes. We show that the 3D surface code has more transversal gates than were previously known. In particular, we demonstrate that certain 3D surface codes have a transversal (non-Clifford) CCZ gate. In Chapter 3, we present results on 3D surface code decoding. Notably, we propose a cellular automaton decoder for correcting bit-flip errors in 3D surface codes with boundaries. We prove that our decoder has an error threshold when stabilizer measurements are perfect and we provide numerical evidence of a threshold when measurements are unreliable. Our decoding results allow us to estimate how good 3D surface codes are at protecting quantum information. We bring the results from previous chapters together in Chapter 4, where we propose fault-tolerant quantum computing architectures based on 3D surface codes that do not require MSD. Finally, in Chapter 5, we discuss our results in a wider context, and suggest directions for future research.

Chapter 2

Transversal gates in three-dimensional surface codes

In most quantum codes, implementing Clifford gates fault-tolerantly is relatively simple and has a low resource cost, but to achieve universality we also need a fault-tolerant implementation of a non-Clifford gate. Unfortunately, costly procedures such as magic state distillation (MSD) are usually required to implement non-Clifford gates fault-tolerantly, as codes that have transversal non-Clifford gates are uncommon. A transversal implementation of a non-Clifford gate is the best fault-tolerant construction we can hope for. This is because transversal operators are depth-one circuits that do not couple qubits in different code blocks, so these operators severely limit the spread of errors. In this Chapter, we show that certain 3D surface codes have a transversal non-Clifford gate. Firstly, we introduce a rectified picture for visualizing stacks of 3D surface codes, which allows us to reason about how stabilizers from different codes overlap. Secondly, we use the rectified picture to show that specific 3D surface codes have a transversal CCZ. Finally, we end the chapter by sketching generalizations of the rectified picture to higher dimensions and curved spaces. Most of the original material in this Chapter previously appeared in [112], a paper by the author and Dan Browne.

2.1 The rectified picture

A rectification (or full truncation) is a geometrical transformation where the vertices of a polyhedron are truncated until the edges of the polyhedron become vertices [113, Chapter 8]. Truncating a polyhedron means cutting off its vertices so that the vertices are replaced by faces. Figure 2.1 shows a series of truncations of a cube,

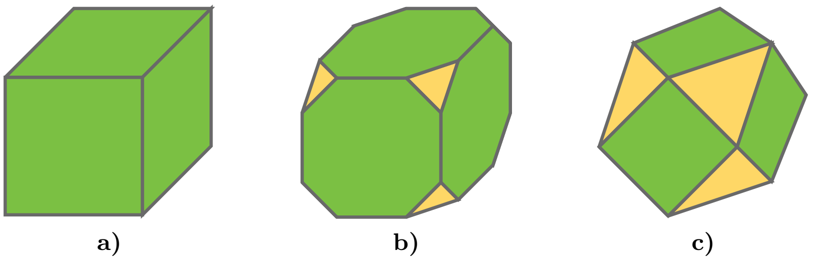


Figure 2.1: Truncating a cube to obtain its rectification. In Figure 2.1b, we show a cube whose vertices have been partially truncated. And in Figure 2.1c, we continue the truncation to obtain a cuboctahedron (rectified cube).

culminating in a rectified cube (cuboctahedron). The rectified picture of 3D surface codes is simply the picture one obtains by rectifying a 3D surface code lattice. This generalizes the well-known rotated picture of 2D surface codes. We begin this section by revising the rotated picture in Section 2.1.1. Then, in Section 2.1.2, we formally introduce the rectified picture. Finally, we define a family of stacked 3D surface codes in Section 2.1.3. This family of stacked codes is one of the main components of the fault-tolerant quantum computing architecture we present in Chapter 4.

2.1.1 The rotated picture

In Chapter 1, we defined 2D surface codes in the standard (Kitaev) picture. We recall that, in this picture, qubits are placed on edges, X-stabilizer generators are associated with vertices and Z-stabilizer generators are associated with faces. There is an equivalent picture where qubits are placed on vertices, which is called the rotated picture. This picture was introduced by Wen [114], and has since been frequently used to analyse 2D surface codes and related codes (e.g. [115, 116, 117, 118]). In the rotated picture, surface codes are defined on four-valent lattices with two-colourable faces [115]. That is, the faces of the lattice can be assigned one of two colours so that no faces that share an edge have the same colour. Consider such a lattice and assume the faces are coloured red (r) and blue (b). We place qubits on the vertices of the lattice, we associate X-stabilizer generators with r-faces and we associate Z-stabilizer generators with b-faces. That is, for each r-face f_r we have a stabilizer,

$$A_{f_r} = \bigotimes_{v \in f_r} X(v), \tag{2.1}$$

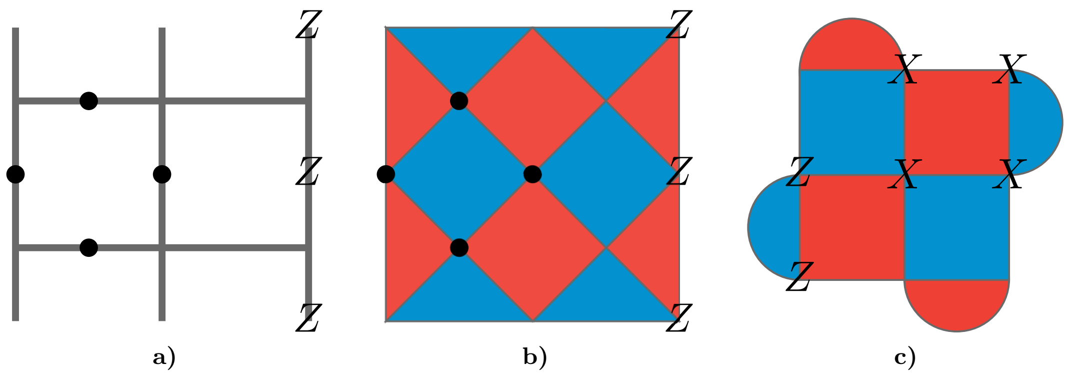


Figure 2.2: The rotated picture of 2D surface codes. In Figure 2.2a and Figure 2.2b, we show the same surface code in the Kitaev picture and the rotated picture, respectively. In each code, we highlight the same subset of qubits (black circles) and the same logical Z-operator. In Figure 2.2c, we show another surface code in the rotated picture, which has the same code distance but fewer physical qubits. We highlight an X-stabilizer (red square) and a Z-stabilizer (blue semicircle).

and for each b-face f_b we have a stabilizer,

$$B_{f_b} = \bigotimes_{v \in f_b} Z(v), \tag{2.2}$$

where $X(v)$ denotes an X-operator acting on the qubit on vertex v, and $Z(v)$ is defined analogously. Figure 2.2 shows a 2D surface code in both the Kitaev picture and the rotated picture.

We also associate colours with the boundaries of rotated picture lattices. We define an r-boundary (b-boundary) as a boundary where all edges on the boundary are part of r-faces (b-faces). With this definition, logical X-operators are strings of X-operators that terminate on different r-boundaries. Similarly, logical Z-operators are strings of Z-operators that terminate on different b-boundaries. By comparing with Section 1.2.1, we see that the b-boundaries (r-boundaries) are rough (smooth) boundaries in the Kitaev picture.

We note that the colours are arbitrary and we could just as well associate the different stabilizer types with the other colours. Therefore, we can use one rotated picture lattice to represent a stack of two 2D surface codes, where we have two qubits at each vertex (one for each code). In one code, we associate X-stabilizer generators with r-faces and Z-stabilizer generators with b-faces, and in the other code we associate X-stabilizer generators with b-faces and Z-stabilizer generators with r-

faces. This is equivalent to two 2D surface codes defined on dual tessellations in the Kitaev picture (see Figure 1.2). We refer to the 2D surface code whose X-stabilizer generators are associated with r-faces (b-faces) as $\mathcal{SC}_r$ ($\mathcal{SC}_b$). We only consider 2D surface codes with one encoded qubit and we denote the logical operators of $\mathcal{SC}_c$ by $\overline{X}_c$ and $\overline{Z}_c$, where $c \in \{r, b\}$.

The rotated picture allows to see that the controlled-Z (CZ) gate is transversal for a stack of two surface codes defined on the same rotated picture lattice. To show this, we need the concept of an operator's vertex support, which we define to be the set of vertices where the qubits in the support of the operator lie. We recall that the support of an operator is the set of qubits on which the operator acts non-trivially. We claim that $U = CZ^{\otimes n}$ is a transversal logical operator, where the CZ gates act on the pairs of (physical) qubits at each of the n vertices of the lattice. Furthermore, we claim that U implements a logical CZ between the encoded qubits of $\mathcal{SC}_r$ and $\mathcal{SC}_b$. To show that this is the case, we compute the action of U on the logical operators of the two codes. The CZ gate transforms Pauli operators in the following way:

$$\begin{aligned} CZ(I \otimes Z)CZ^\dagger &= I \otimes Z, \\ CZ(Z \otimes I)CZ^\dagger &= Z \otimes I, \\ CZ(I \otimes X)CZ^\dagger &= Z \otimes X, \\ CZ(X \otimes I)CZ^\dagger &= X \otimes Z. \end{aligned} \tag{2.3}$$

We now show that U implements the unitary defined in Equation 2.3 at the logical level. First, consider how U transforms logical identity operators (stabilizers). From Equation 2.3, it is clear that U has no effect on the Z-stabilizers of either code but it does affect the X-stabilizers. Remembering that X-stabilizers in one code have the same vertex support as Z-stabilizers in the other code, we conclude that U maps an X-stabilizer of one code to a tensor product of the same X-stabilizer and a Z-stabilizer of the other code with the same vertex support. Overall, we see that U maps stabilizers to stabilizers and therefore maps the logical identity to the logical identity. Next, we consider the transformation of the logical Pauli operators. As the logical Z-operators of both codes are tensor products of Z-operators acting on the physical qubits, Equation 2.3 immediately tells us that these logical operators are left invariant by U. The logical X-operators of both codes are tensor products of X-operators acting on the physical qubits, and each logical X-operator of one code has the same vertex support as a logical Z-operator of the other code. Therefore, U maps every $\overline{X}_r$ to some $\overline{X}_r \otimes \overline{Z}_b$ and U maps every $\overline{X}_b$ to some $\overline{X}_b \otimes \overline{Z}_r$. This concludes our proof that U is a transversal implementation of the CZ gate.

2.1.2 Rectifying 3D surface codes

We are now ready to formally introduce the rectified picture of 3D surface codes. We begin by explaining how this picture relates to the standard definition of 3D surface codes. Then, we show that the rectified picture is natural for understanding how the stabilizers of different codes overlap in a stack of 3D surface codes.

Consider a 3D surface code lattice in the standard (Kitaev) picture. We recall that, in this picture, qubits are placed on edges, X-stabilizer generators are associated with vertices, and Z-stabilizer generators are associated with faces. To obtain a rectified-picture lattice, we rectify every cell of the original lattice. As we noted at the beginning of Section 2.1, to rectify a polyhedron we truncate its vertices until its edges become vertices (see Figure 2.1). Under rectification, the elements of a lattice transform as follows: edges are mapped to vertices, cells and vertices are mapped to cells, and faces are mapped to faces. Therefore, in the rectified picture, qubits are on vertices, X-stabilizer generators are associated with cells and Z-stabilizer generators are associated with faces. Figure 2.3 shows a 3D surface code lattice in the Kitaev picture and in the rectified picture.

There is an analogous transformation that maps 3D surface code dual lattices (with qubits on faces, see Section 1.2.1) to rectified-picture lattices. This transformation is called a face-rectification and is equivalent to taking the dual of every cell in the lattice. To construct the dual of a polyhedron, we create vertices at the centre of its faces and we connect these vertices with edges if their corresponding faces in the original polyhedron share an edge. Figure 2.4 shows two polyhedra and their duals. Finally, we note that the rectified picture is similar to the primal lattice picture of 3D colour codes [94] and much of our terminology was inspired by these codes.

The utility of the rectified picture comes when we consider stacks of 3D surface codes i.e. different codes occupying the same physical space. This is because different

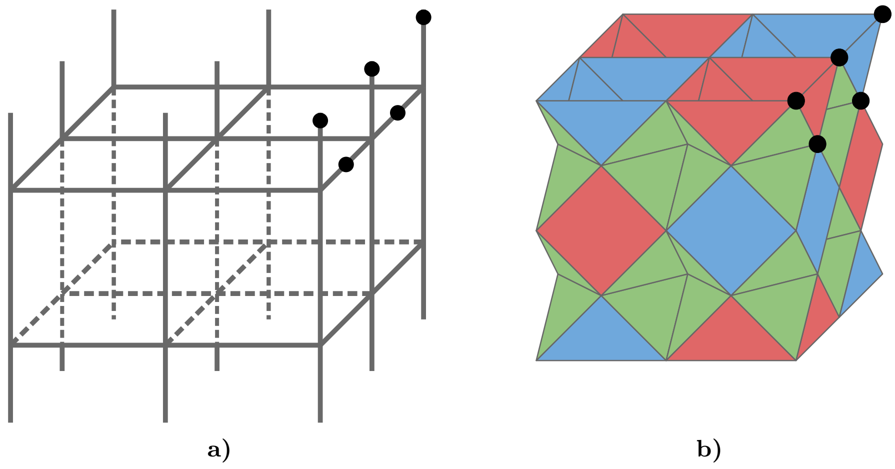


Figure 2.3: The same 3D surface code lattice shown in the Kitaev picture (Figure 2.3a) and the rectified picture (Figure 2.3b). We highlight the same subset of qubits (black circles) in each picture. Figure adapted from Vasmer and Browne [112].

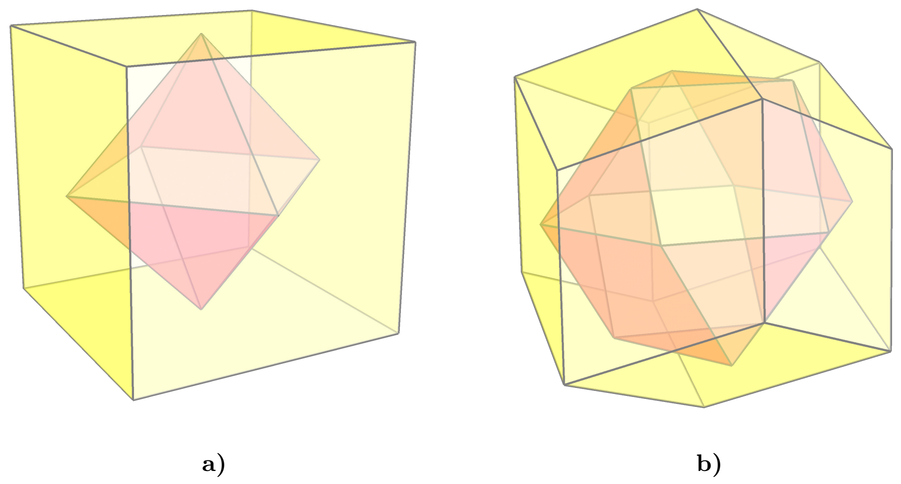


Figure 2.4: Polyhedra and their duals. Figure 2.4a shows a cube and its dual, an octahedron. Figure 2.4b shows a rhombic dodecahedron (yellow) and its dual, a cuboctahedron (red). Figure adapted from Vasmer and Browne [112].

lattices in the Kitaev picture correspond to the same lattice in the rectified picture. We now illustrate this with an example. Consider a 3D surface code defined on a cubic lattice. For simplicity, we consider a tessellation of the 3-torus by cubes. A 3-torus is a manifold that can be constructed by identifying the opposite facets of a cube. Let us examine how the cubic lattice transforms under rectification. The vertices of the lattice are mapped to octahedra, the edges are mapped to vertices,

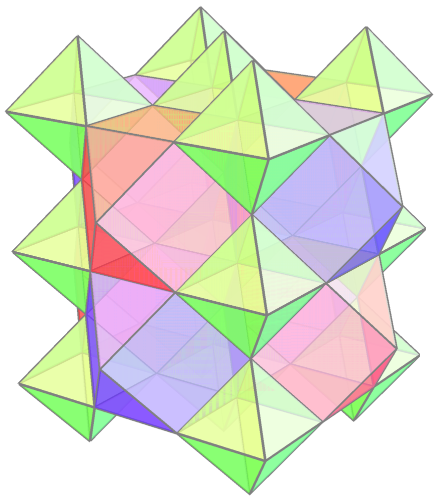


Figure 2.5: The rectified cubic lattice. Octahedra are green and cuboctahedra are red and blue. Figure adapted from Vasmer and Browne [112].

the square faces are mapped to square faces and the cubic cells are mapped to cuboctahedra (the polyhedron shown in Figure 2.1c). In the resultant lattice, two octahedra and four cuboctahedra meet at every vertex. This lattice, which is shown in Figure 2.5, is usually called a rectified cubic lattice.

Next, we show that another lattice has the same rectification of the cubic lattice. Consequently, we can use a single rectified lattice to understand a stack of 3D surface codes defined on different lattices. We consider a lattice where six octahedra and eight tetrahedra meet at every vertex. This lattice is usually called a tetrahedral-octahedral lattice. Again, we restrict ourselves to a tessellation of a 3-torus for simplicity. Under rectification, the vertices of the lattice are mapped to cuboctahedra, the triangular faces are mapped to triangular faces, the tetrahedral cells are mapped to octahedra and the octahedra are mapped to cuboctahedra. The resultant lattice is also a rectified cubic lattice.

We can arrange one cubic lattice and two tetrahedral-octahedral lattices so that all three lattices are transformed into exactly the same lattice under rectification. To see how this works, it is easiest to consider the dual lattices and work with the face-rectification transformation. To construct the dual of a 3D lattice, we create vertices at the centre of all the cells and connect these vertices if the corresponding cells

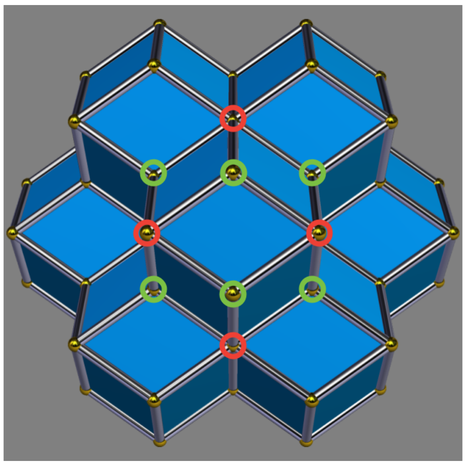


Figure 2.6: The rhombic-dodecahedral lattice (dual to the tetrahedral-octahedral lattice). We highlight the acute vertices (red) and obtuse vertices (green) of one rhombic dodecahedron. This image was generated using Great Stella [119].

share a face. The cubic lattice is self-dual while the dual lattice of the tetrahedral-octahedral lattice is the rhombic-dodecahedral lattice, a portion of which is shown in Figure 2.6. The rhombic-dodecahedral lattice is a tessellation where every cell is a rhombic dodecahedron (see Figure 2.4b). There are two types of vertices in the lattice: acute vertices and obtuse vertices. Four rhombic dodecahedra meet at obtuse vertices and six rhombic dodecahedra meet at acute vertices.

Let us define a coordinate system for a 3-torus with length $L \in 2\mathbb{Z}^+$. We use a standard Cartesian system where the coordinates are (i,j,k) for $i,j,k \in \mathbb{Z}/L\mathbb{Z}$. The vertices of the cubic lattice lie at these coordinates, which we divide into two sets: the coordinates with even sum and the coordinates with odd sum. The centres of the cubes have half-integer coordinates. We arrange one of the rhombic-dodecahedral lattices in such a way that the acute vertices of the rhombic dodecahedra are at the even coordinates. In this arrangement, the obtuse vertices of the rhombic dodecahedra are at the centres of the cubes and the rhombic dodecahedra are centred at odd coordinates. We illustrate this arrangement in Figure 2.7a. We place the second rhombic-dodecahedral lattice so that the acute vertices of this lattice are at the odd coordinates. Consequently, the obtuse vertices of this lattice are at the centres of cubes and the rhombic dodecahedra are centred at even vertices. The arrangement of the three lattices is shown in Figure 2.7b.

In the arrangement we have just described, all three lattices are mapped to an

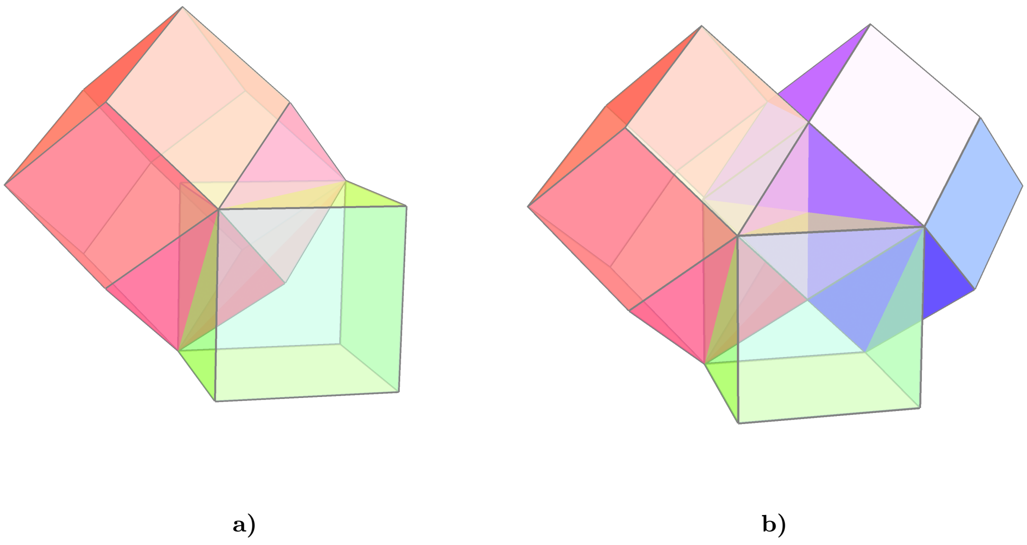


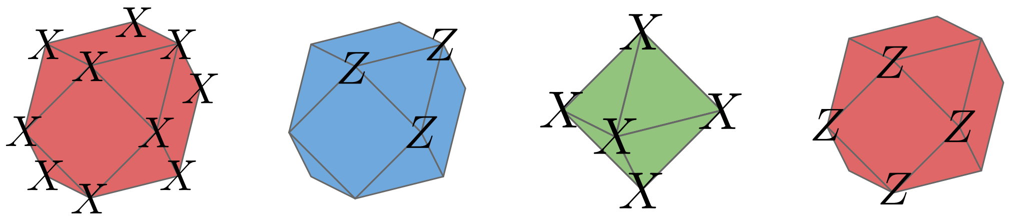
Figure 2.7: Arranging a cubic lattice (green) and two rhombic-dodecahedral lattices (red and blue) so that their face-rectifications are identical. We show a single cell from each lattice. Figure adapted from Vasmer and Browne [112].

identical rectified cubic lattice by the face-rectification transformation. To see why this is true, we compare how the different lattices transform at the same coordinates. First, we review how the different lattices transform. Under face-rectification, the cells of the cubic lattice transform into octahedra and the vertices transform into cuboctahedra. The cells and acute vertices of the rhombic-dodecahedral lattice transform into cuboctahedra, and the obtuse vertices transform into octahedra. As the obtuse vertices of rhombic dodecahedra lie at the centres of cubes, all three lattices transform the same way at these points. Similarly, because the vertices of cubes and acute vertices of one of the rhombic-dodecahedral lattices both lie at the centres of the cells of the other rhombic-dodecahedral lattice, the three lattices transform in the same way at these points. Therefore the three lattices transform identically at every coordinate.

Now that we have shown that three Kitaev picture lattices correspond to one rectified-picture lattice, we detail the stabilizer generators of a stack of three codes in the rectified picture. We place three qubits at each vertex (one per code) of a rectified cubic lattice that tessellates the 3-torus. The cells of this lattice are three-colourable i.e. we can assign each cell one of three colours so that cells that share a face have different colours. The colours have no physical significance, but we use them to define the stabilizer groups of the three codes. We choose a colouring where octahedra are green (g) and the two sets of cuboctahedra are red (r) and blue (b) (see Figure 2.5). We assign each face of the lattice the colour of the two cells it is part of. For example, a face shared by a r-cuboctahedron and a g-octahedron is a rg-face. We use the colours to label the three codes, $\mathcal{SC}_c$ for $c \in \{r, g, b\}$. We associate the X-stabilizer generators of $\mathcal{SC}_c$ with c-cells and we associate the Z-stabilizer generators of $\mathcal{SC}_c$ with $c'c''$-faces. Table 2.1 and Figure 2.8 detail the stabilizer generators of the three codes in the stack.

Code	X stabilizers	Z stabilizers
$\mathcal{SC}_r$	r-cuboctahedra	bg-faces (triangles)
$\mathcal{SC}_b$	b-cuboctahedra	rg-faces (triangles)
$\mathcal{SC}_g$	g-octahedra	rb-faces (squares)

Table 2.1: The stabilizer generators of a stack of three 3D surface codes defined on a rectified cubic lattice.



a) $\mathcal{SC}_r$ X-stabilizer **b)** $\mathcal{SC}_r$ Z-stabilizer **c)** $\mathcal{SC}_g$ X-stabilizer **d)** $\mathcal{SC}_g$ Z-stabilizer

Figure 2.8: The stabilizer generators of $\mathcal{SC}_r$ and $\mathcal{SC}_g$. The stabilizer generators of $\mathcal{SC}_b$ are identical to those of of $\mathcal{SC}_r$ except with red and blue interchanged. The maximum stabilizer weight in $\mathcal{SC}_g$ is six and the the maximum stabilizer weight in $\mathcal{SC}_r$ and $\mathcal{SC}_b$ is twelve. Figure adapted from Vasmer and Browne [112].

Given the definitions of the stabilizer generators in Table 2.1, we can easily understand how stabilizers from different codes overlap. For example, we immediately observe that X-stabilizer generators from two different codes overlap on a Z-stabilizer of the third code. This would not be obvious if we considered three interlocking lattices in the Kitaev picture. We note that this overlap property is similar to the tri-orthogonality property we discussed in Section 1.3.1, except now we have three codes instead of one. In Section 2.2, we use the overlap of stabilizers in our proof that CCZ is transversal for a stack of three 3D surface codes. But first, in the next section, we introduce a family of 3D surface codes defined on rectified cubic lattices with boundaries.

2.1.3 A family of 3D surface codes with boundaries

If we want to use 3D topological codes in a fault-tolerant quantum computing architecture, then defining our codes on tessellations of the 3-torus is not the most practical choice. One obvious problem is that the 3-torus cannot be embedded in 3D space. In this section, we define a family of 3D surface codes defined on lattices that tessellate the cube, and we verify the properties of the codes.

We consider rectified cubic lattices with boundaries and we use the same colouring of the cells as we used in Section 2.1.2. On the boundaries, we assign faces that are part of only one cell the colour they would have in an infinite lattice. Because of the regularity of the rectified cubic lattice, we can always imagine a lattice with boundaries to be a restricted region of the infinite lattice. The lattices in our family have two types of boundary. One type of boundary slices a layer of cuboctahedra in half and the other type of boundary slices between a layer of cuboctahedra. We call these boundaries half-cuboctahedra boundaries and full-cuboctahedra boundaries, respectively (see the boundaries of the lattice in Figure 2.3b). Each lattice in our family is a tessellation of the cube, with two half-cuboctahedron boundaries and four full-cuboctahedron boundaries. Opposite boundaries are the same type. We parameterize the lattices in our family with an integer, $d \geq 2$, which will ultimately be equal to the code distance of the 3D surface codes supported on the corresponding lattice. Figure 2.9 shows the $d = 3$ lattice.

We specify the micro-structure of a distance d lattice by dividing it into 2D layers, each of which is parallel to the half-cuboctahedra boundaries. There are two types of layer in this division, which we call "chequerboard layers" and "diamond layers", due to their appearance. Figure 2.10 shows the structure of both types of layer in the $d = 3$ lattice and the $d = 4$ lattice. In a distance d lattice, there are d chequerboard layers and $(d-1)$ diamond layers and the two types of layer alternate. The half-cuboctahedra boundaries are themselves chequerboard layers. Layers directly above and below each other are connected by edges according to the structure of the rectified cubic lattice (see Figure 2.9).

Next, we describe and verify the properties of the three 3D surface codes defined

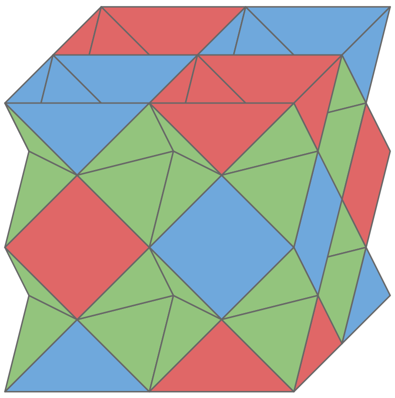


Figure 2.9: The $d=3$ rectified cubic lattice with boundaries. The top and bottom boundaries are half-cuboctahedra boundaries and the other four boundaries are full-cuboctahedra boundaries. Figure adapted from Vasmer and Browne [112].

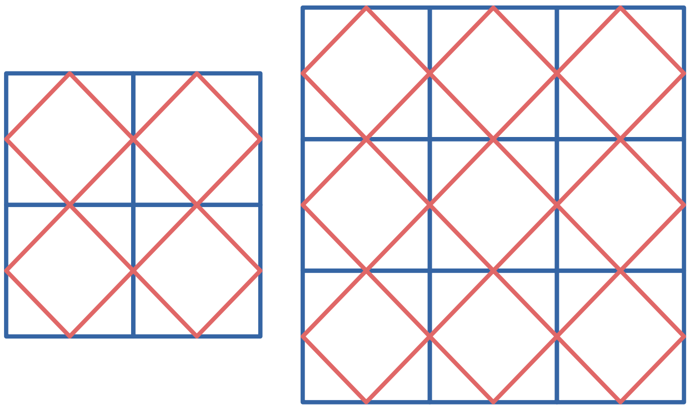


Figure 2.10: The two types of layer in a $d=3$ (left) and a $d=4$ (right) rectified cubic lattice with boundaries. Chequerboard layers are shown in blue and diamond layers are shown in red. Figure adapted from Vasmer and Browne [112].

on a distance d lattice. We place three qubits at each vertex of the lattice (one qubit per code). Each chequerboard layer in the lattice has d^2 vertices and each diamond layer has $2d(d-1)$ vertices. Therefore, the number of physical qubits in each code is

$$n = d^3 + 2d(d-1)^2 = 3d^3 - 4d^2 + 2d. \tag{2.4}$$

As in Section 2.1.2, we label each code, $\mathcal{SC}_c$, by the colour of its X-stabilizer generators. Specifically, we associate the X-stabilizer generators of $\mathcal{SC}_c$ with c-cells and we associate the Z-stabilizer generators of $\mathcal{SC}_c$ with $c'c''$-faces, where $c, c', c'' \in \{r, g, b\}$.

Table 2.1 and Figure 2.8 show the stabilizer generators of the three codes. We denote the X-stabilizer group of $\mathcal{SC}_c$ by $\mathcal{S}_X^c$ and we denote the Z-stabilizer group of $\mathcal{SC}_c$ by $\mathcal{S}_Z^c$.

We also associate colours with the boundaries of our rectified cubic lattices. We define a c-boundary to be a boundary that corresponds to a rough boundary in $\mathcal{SC}_c$ and a smooth boundary in $\mathcal{SC}_{c'}$ and $\mathcal{SC}_{c''}$. Since the underlying Kitaev picture lattices are regular, we can use a simple definition of the rough and smooth boundaries, which is equivalent to the definition in terms of excitations we used in Section 1.2.1. In all three codes in the stack, each qubit in the bulk of the lattice is in the support of two X-stabilizer generators and four Z-stabilizer generators. We define a rough boundary to be a boundary of the lattice where every qubit is in the support of a single X-stabilizer generator and four Z-stabilizer generators. Similarly, we define a smooth boundary to be a boundary of the lattice where every qubit is in the support of two X-stabilizer generators and fewer than four Z-stabilizer generators.

If we want stacks of codes in our family to have a transversal CCZ gate, we need to make sure that the logical operators of different codes have the correct overlap (see Section 2.2). It transpires that this requirement will be satisfied if the lattices in our family have two boundaries of each colour where opposite boundaries have the same colour. The half-cuboctahedra boundaries in a distance d lattice are automatically valid g-boundaries. However, the full-cuboctahedra boundaries are neither r-boundaries nor b-boundaries. The problem is that the four full-cuboctahedra boundaries are identical. Therefore, we need to break the symmetry between the four full-cuboctahedra boundaries in just the right way. We do this by adding additional low weight stabilizers with support on full-cuboctahedra boundaries to two of the codes. These stabilizers are analogous to the weight-two stabilizers on the boundaries of the 2D surface code shown in Figure 2.2c. In Figure 2.11, we show the additional stabilizers we add to $\mathcal{SC}_r$ and $\mathcal{SC}_b$ to ensure that the full cuboctahedra boundaries have the correct structure.

We now show that with the additional stabilizers shown in Figure 2.11, the three codes have the desired boundaries. That is, the lattice has two boundaries of each colour and opposite boundaries are the same colour. We recall that c-

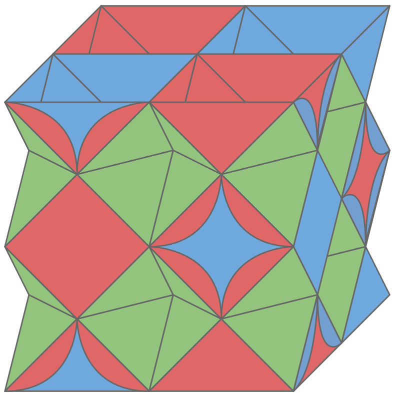


Figure 2.11: Low weight stabilizers on the boundaries of the lattice. On the front (full-cuboctahedra) boundary, we associate $\mathcal{SC}_r$ X-stabilizers with the faces of the b-cuboctahedra (blue faces). In addition, we associate $\mathcal{SC}_b$ Z-stabilizers with some of the edges of these blue faces (red circular segments). These edges would have been part of rg-faces in the infinite lattice. In effect, we have added multiple 2D "flattenings" of r-cuboctahedra to the lattice. The edges of these 2D flattenings are themselves 1D flattenings of rg-faces. We add analogous stabilizers to the back boundary. Next, consider the left and right boundaries. We associate $\mathcal{SC}_b$ X-stabilizers with the faces of the r-cuboctahedra (red faces) on these boundaries. We also associate $\mathcal{SC}_r$ Z-stabilizers with some of the edges of these faces (blue circular segments). Figure adapted from Vasmer and Browne [112].

boundaries are defined to be rough boundaries in $\mathcal{SC}_c$ and smooth boundaries in $\mathcal{SC}_{c'}$ and $\mathcal{SC}_{c''}$. First consider top and bottom boundaries in Figure 2.11. We claim these boundaries are g-boundaries. Each vertex on the top/bottom boundaries is a member of a single g-octahedron ($\mathcal{SC}_g$ X-stabilizer). Each vertex is also a member of four rb-faces ($\mathcal{SC}_g$ Z-stabilizers), except where a top/bottom boundary meets one of the other boundaries. The top/bottom boundaries are, therefore, rough boundaries in $\mathcal{SC}_g$. Each vertex on the top/bottom boundaries is a member of two r-cuboctahedra (including 2D flattenings shown in Figure 2.11) and two b-cuboctahedra (including 2D flattenings), except where the top/bottom boundaries meet a left/right boundary or a front/back boundary, respectively. The vertices on the top/bottom boundaries are all members of fewer than four rg-faces (including 1D flattenings shown in Figure 2.11) and fewer than four bg-faces (including 1D flattenings). Therefore, the top/bottom boundaries are smooth boundaries in $\mathcal{SC}_b$ and $\mathcal{SC}_r$.

The next pair of boundaries we consider are the front and back boundaries in Figure 2.11. We claim that these boundaries are b-boundaries. Due to the additional stabilizers shown in Figure 2.11, each vertex on the front/back boundaries is a member of two r-cuboctahedra (including 2D flattenings) and two g-octahedra, except where the front/back boundaries meet the left/right boundaries and top/bottom boundaries, respectively. However, each vertex on the front/back boundaries is a member of a single b-cuboctahedron. Every vertex on the front/back boundaries is a member of fewer than four rb-faces and fewer than four bg-faces (including 1D flattenings). But each vertex is a member of four rg-faces (including 1D flattenings) except for the vertices that are also on one of the other boundaries. Therefore, the front/back boundaries are rough boundaries in $\mathcal{SC}_b$ and smooth boundaries in $\mathcal{SC}_r$ and $\mathcal{SC}_g$, as claimed. The argument for the left and right boundaries being r-boundaries is identical to the argument for the b-boundaries, except with r and b exchanged.

Next, we show that each of the three codes has one logical qubit. The number of logical qubits in a stabilizer code is equal to the number of physical qubits minus the number of independent stabilizer generators. So we need to count the stabilizer generators of each of the three codes. We begin with $\mathcal{SC}_g$. In this code, X-stabilizer generators are associated with g-cells (octahedra) and Z-stabilizer generators are associated with rb-faces. Consider the top g-boundary of a distance d lattice oriented the same way as the lattice in Figure 2.11. This boundary has the structure of a chequerboard layer and each vertex on this boundary is a member of a single (complete or incomplete) octahedron. Chequerboard layers have d^2 vertices so we have d^2 octahedra that are situated directly below the top boundary. Every other chequerboard layer (except the bottom layer) also has d^2 octahedra situated below it. There are d chequerboard layers so there are $d^2(d-1)$ octahedra in a distance d lattice. The X-stabilizers we associate with these octahedra are all independent. Therefore, the number of independent X-stabilizer generators in $\mathcal{SC}_g$, i.e. the rank of $\mathcal{S}_X^g$, is

$$\operatorname{rank}(\mathcal{S}_X^g) = d^2(d-1). \tag{2.5}$$

We now count the Z-stabilizer generators of $\mathcal{SC}_g$. As we stated previously, these stabilizers are associated with the rb-faces of the lattice. We split these faces into two groups: faces that are parallel to g-boundaries, and faces that are parallel to the r-boundaries or the b-boundaries. In a distance d lattice, we have $(d-1)^2$ rb-faces parallel to the g-boundaries in each diamond layer. There are $d-1$ diamond layers, so there are $(d-1)^3$ rb-faces parallel to the g-boundaries. Each chequerboard layer cuts through $2d(d-1)$ rb-faces that are parallel to the r-boundaries or the b-boundaries. There are d chequerboard layers, so there are $2d^2(d-1)$ of these rb-faces. Therefore, the total number of rb-faces in a distance d lattice is $(d-1)(3d^2-2d+1)$. However, these stabilizers are not all independent. We can multiply the Z-stabilizers associated with the rb-faces of any cuboctahedron (both full cuboctahedra and half cuboctahedra) to get the identity. Consequently, we must remove one Z-stabilizer from the list of independent stabilizer generators for every cuboctahedron in the lattice. Each chequerboard layer has $(d-1)^2$ cuboctahedra and there are d chequerboard layers, so in total we have $d(d-1)^2$ cuboctahedra in a distance d lattice. Subtracting the redundant stabilizers, we have

$$\mathrm{rank}(\mathcal{S}_Z^g) = (d-1)(2d^2-d+1). \tag{2.6}$$

Combining Equations 2.5 and 2.6, we see that the total number of independent stabilizer generators in $\mathcal{SC}_g$ is

$$\begin{aligned}\mathrm{rank}(\mathcal{S}_X^g)+\mathrm{rank}(\mathcal{S}_Z^g) &= (d-1)(3d^2-d+1),\\ &= 3d^3-4d^2+2d-1.\end{aligned} \tag{2.7}$$

By comparing Equations 2.7 and 2.4, we see that $\mathcal{SC}_g$ has has $n-1$ independent stabilizer generators, where n is the number of physical qubits in the code. Therefore, $\mathcal{SC}_g$ encodes a single logical qubit.

Next, we count the stabilizer generators of $\mathcal{SC}_b$. The X-stabilizer generators of this code are associated with b-cells (including the 2D flattenings) and the Z-stabilizer generators are associated with rg-faces (including the 1D flattenings). First, we count the X-stabilizer generators of $\mathcal{SC}_b$. Consider the chequerboard layers parallel to the g-boundaries. Each chequerboard layer has $(d-1)^2$ cuboctahedra (half of which are r and half of which are b). There are d chequerboard layers, so there are $d(d-1)^2/2$ $\mathcal{SC}_b$ X-stabilizers associated with b-cuboctahedra (either full cuboctahedra or half cuboctahedra). Now consider the r-boundaries of the lattice. On each r-boundary we have additional $\mathcal{SC}_b$ X-stabilizers associated with the faces of r-cuboctahedra (as explained in Figure 2.11). There are $d(d-1)$ of these faces in a distance d lattice so we have $d(d-1)$ additional $\mathcal{SC}_b$ X-stabilizers. The stabilizers we have just detailed are all independent. Hence, the rank of $\mathcal{S}_X^b$ is

$$\operatorname{rank}(\mathcal{S}_X^b) = \frac{(d-1)}{2}(d^2+d). \tag{2.8}$$

We now count the Z-stabilizer generators of $\mathcal{SC}_b$. These stabilizers are associated with rg-faces (and their 1D flattenings). The rg-faces are part of r-cuboctahedra, which we counted in the previous paragraph. The $(d-1)^2/2$ half r-cuboctahedra on the g-boundaries each have four rg-faces. The chequerboard layers that are parallel to the g-boundaries but are not the g-boundaries each have $(d-1)^2/2$ full r-cuboctahedra with eight rg-faces. There are d chequerboard layers in a distance d lattice and two of these layers are the g-boundaries. Therefore, the total number of $\mathcal{SC}_b$ Z-stabilizers associated with rg-faces is $4(d-1)^3$. As shown in Figure 2.11, we also associate $\mathcal{SC}_b$ Z-stabilizers with the edges of the faces that belong to b-cuboctahedra on the b-boundaries. These faces are either square or triangular. Each square face has three independent Z-stabilizers associated with its edges and each triangular face has two independent Z-stabilizers associated with its edges. There are $2(d-1)$ such triangular faces and $(d-1)(d-2)$ such square faces on the b-boundaries. Therefore, the total number of independent weight-two Z-stabilizer generators in $\mathcal{SC}_b$ is $(d-1)(3d-2)$.

Some of the Z-stabilizers we have counted so far are not independent. Consider a complete g-octahedron. Half of its faces are rg-faces and half are bg-faces. The product of the Z-stabilizers associated with the rg-faces is the identity, as each vertex is part of exactly two rg-faces. The product of all the Z-stabilizers associated with the rg-faces of each complete r-cuboctahedron is also the identity for the same reason. Therefore we must remove a single Z-stabilizer from the list of independent stabilizer generators for each complete octahedron and r-cuboctahedron. Every chequerboard layer parallel to the g-boundaries (except the bottom g-boundary) has a complete octahedron below all the vertices in the bulk of the layer. There are therefore $(d-1)(d-2)^2$ complete octahedra in a distance d lattice. We have

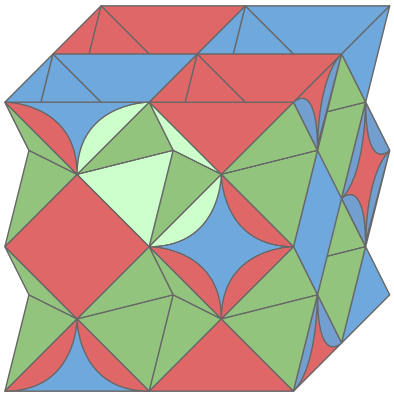


Figure 2.12: Redundant Z-stabilizers in $\mathcal{SC}_b$. We can construct the identity by multiplying the Z-stabilizers associated with the rg-faces and edges of half octahedra on the b-boundaries. We have highlighted one such collection of faces and edges in bright green. Figure adapted from Vasmer and Browne [112].

already counted the $(d-1)^2(d-2)/2$ complete r-cuboctahedra. There is also one other redundancy we have not taken into account. We can construct the identity by multiplying the Z-stabilizers associated with the rg-faces and edges of the half octahedra on the b-boundaries, as illustrated in Figure 2.12. There are $2(d-1)(d-2)$ of these half octahedra. In total we need to remove $(d-1)(3d^2-7d+2)/2$ redundant Z-stabilizers from the list of independent stabilizer generators. The rank of $\mathcal{S}_Z^b$ is therefore

$$\begin{aligned} \operatorname{rank}(\mathcal{S}_Z^b) &= \frac{(d-1)}{2}(8(d-1)^2+6d-4-3d^2+7d-2), \\ &= \frac{d-1}{2}(5d^2-3d+2). \end{aligned} \tag{2.9}$$

Summing Equations 2.8 and 2.9 gives us the number of independent stabilizer generators in $\mathcal{SC}_b$:

$$\begin{aligned} \operatorname{rank}(\mathcal{S}_X^b)+\operatorname{rank}(\mathcal{S}_Z^b) &= (d-1)(3d^2-d+1), \\ &= 3d^3-4d^2+2d-1. \end{aligned} \tag{2.10}$$

By comparing Equations 2.10 and 2.4, we see that $\mathcal{SC}_b$ has has $n-1$ independent stabilizer generators, where n is the number of physical qubits in the code. Therefore, $\mathcal{SC}_b$ encodes a single logical qubit. $\mathcal{SC}_r$ also encodes a single logical qubit by the

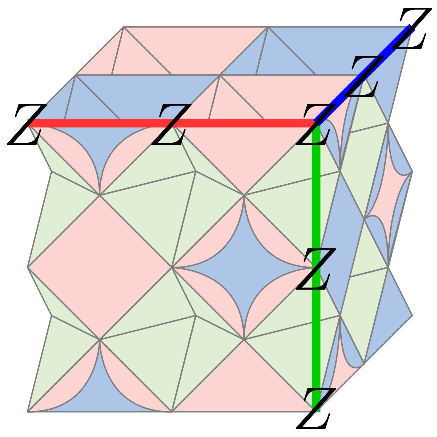


Figure 2.13: Canonical $\overline{Z}_r$ (red line), $\overline{Z}_g$ (green line) and $\overline{Z}_b$ (blue line) operators. The canonical $\overline{X}_c$-operators act on every qubit on one of the c-boundaries. Figure adapted from Vasmer and Browne [112].

same argument, except with r and b swapped everywhere.

To finish our discussion of rectified-cubic codes with boundaries, we detail the logical operators of the three codes. In $\mathcal{SC}_c$, $\overline{Z}_c$-operators are strings of Z-operators from one c-boundary to the other and $\overline{X}_c$-operators are membranes of X-operators with a boundary that spans the c' and c''-boundaries. It is useful to define a canonical set of logical operators for each code. The canonical $\overline{Z}_c$-operators lie along the lines where c'-boundaries meet c''-boundaries. That is, given a c'-boundary and a c''-boundary that share vertices, a canonical $\overline{Z}_c$-operator acts on all qubits that are members of both boundaries. Figure 2.13 shows example canonical $\overline{Z}_c$-operators for the three codes in a single stack. These canonical $\overline{Z}_c$-operators are all weight d, where d is the integer that parameterizes the lattice. We define the canonical $\overline{X}_c$-operators as membranes of X-operators that act on every qubit on one of the c-boundaries. The canonical $\overline{X}_g$-operators are weight d^2 and the canonical $\overline{X}_r$ and $\overline{X}_b$-operators are weight $d^2 + (d-1)^2$.

In this section, we introduced the rectified picture of 3D surface codes and we used our new picture to better understand the structure of stacks of 3D surface codes. We also defined a family of 3D surface codes with boundaries. In the next section, we make use of the rectified picture to show that our family of codes has two transversal non-Pauli gates, one of which is non-Clifford.

2.2 Transversal CCZ

In this section, we prove that the CCZ gate is transversal for triples of 3D surface codes with certain properties. Our proof builds on previous proofs of the transversality of non-Clifford gates in tri-orthogonal codes [71, 72, 75, 74, 76]. We recall that a non-Clifford gate such as the CCZ gate can be combined with the Clifford group to make a universal gate set. In 2D surface code architectures, implementing a non-Clifford gate has a relatively large overhead. In contrast, transversal gates have the lowest possible overhead because they are depth-one circuits and they do not use ancilla qubits.

We consider a stack of three 3D surface codes defined on the same rectified-picture lattice. For the sake of clarity, we restrict ourselves to the case where each code has a single logical qubit, though the proof can be generalized to the case where the codes have multiple logical qubits. We denote the three codes by $\mathcal{SC}_r$, $\mathcal{SC}_b$ and $\mathcal{SC}_g$. In each code we chose a canonical $\overline{X}_c$-operator. Let $H_c \in \mathbb{F}_2^{m\times n}$ be the $m \times n$ binary parity check matrix of the X-stabilizer generators of $\mathcal{SC}_c$, where n is the number of physical qubits in the code (the same for all three codes), m is the number of X-stabilizer generators (which could be different for different codes) and $c \in \{r,g,b\}$. That is, $(H_c)_{ij} = 1$ if the X-stabilizer corresponding to row i acts non-trivially on qubit j and $(H_c)_{ij} = 0$ otherwise. We denote the row span of H_c by G_c^0. Let $X_c \in \mathbb{F}_2^n$ by a n-bit vector that describes the support of $\overline{X}_c$. That is, X_c has a 1 at position j if, and only if, $\overline{X}_c$ acts non-trivially on the qubit at the vertex corresponding to j. We define the coset $G_c^1 = \{X_c + g : g \in G_c^0\}$. With these definitions, we can write the encoded state of $\mathcal{SC}_c$ as follows:

$$|\overline{\alpha}\rangle_c = \frac{1}{\sqrt{|G_c^\alpha|}} \sum_{g\in G_c^\alpha} |g\rangle_c, \tag{2.11}$$

where the c subscript on the kets refers to the qubits of $\mathcal{SC}_c$, $|G_c^\alpha|$ is the number of elements in G_c^α and $\alpha \in \{0,1\}$. This form of the encoded state can be derived from applying the X-stabilizer projector to the state $|0\rangle^{\otimes n}$. Such a state is clearly a $+1$ eigenstate of the X-stabilizers and will also be a $+1$ eigenstate of the Z-stabilizers due to the stabilizer group being Abelian.

We define the overlap of logical operators (including stabilizers) from different codes to be the vertices of the lattice where all of the logical operators act non-trivially. And we define the vertex support of a logical operator to be the set of vertices that host the qubits in the support of the logical operator. To prove our main theorem, we need the following Lemma.

Lemma 1 *Given a finite set of binary vectors $\{a_j\}$, the parity of their sum is equal to the sum of their parities.*

This lemma is easy to prove. For completeness, we include a proof in Appendix A. In addition, we need the the following pair of Lemmas, which concern the overlap of stabilizers and logical operators.

Lemma 2 *Consider a stack of three 3D surface codes defined on the same rectified-picture lattice. Suppose that the overlap of any two X-stabilizer generators of different codes is equal to the vertex support of a Z-stabilizer of the third code. Then, the overlap of any two X-stabilizers from different codes is equal to the vertex support of a Z-stabilizer of the third code.*

Proof. Let $t \in G_r^0$ denote the support of an X-stabilizer of $\mathcal{SC}_r$ and let $u \in G_g^0$ denote the support of an X-stabilizer of $\mathcal{SC}_g$. We can decompose each of these operators into sums of vectors denoting the support of stabilizer generators, i.e. $t = \sum_i t_i$ and $u = \sum_j u_j$, where t_i and u_j are rows of H_r and H_g, respectively. The overlap of these two operators is given by the following formula:

$$t \circ u = \sum_i t_i \circ \sum_j u_j = \sum_{ij} t_i \circ u_j, \tag{2.12}$$

where $t \circ u$ denotes the bit-wise binary product between t and u. By assumption, $t_i \circ u_j$ is equal to the vertex support of a Z-stabilizer of $\mathcal{SC}_b$ for any i and j. And because the set of Z-stabilizers form a group, $\sum_{ij} t_i \circ u_j = t \circ u$ is also equal to the vertex support of a Z-stabilizer of $\mathcal{SC}_b$. An identical argument holds for any pair of X-stabilizers of different codes. □

Lemma 3 *Consider a stack of three 3D surface codes defined on the same rectified-picture lattice, where each code has one encoded qubit. Assume that the overlap of any two X-stabilizer generators of different codes is equal to the vertex support of a Z-stabilizer of the third code. Suppose that we can find three $\overline{X}$-operators (one from each code) that have the following properties:*

i The overlap of any two of the $\overline{X}$-operators and the X-stabilizer group of the third code is even.

ii The overlap of the three $\overline{X}$-operators is odd.

Then, any three logical X-operators (one from each code) also have the same properties.

Proof. Let X_r, X_g, and X_b denote the support of $\overline{X}_r$, $\overline{X}_g$ and $\overline{X}_b$-operators that satisfy the requirements of the Lemma. The parity of the overlap of an X-stabilizer of $\mathcal{SC}_b$ with aribtrary logical X-operators of $\mathcal{SC}_r$ and $\mathcal{SC}_g$ is equal to

$$|(X_r + t) \circ (X_g + u) \circ v| \mod 2, \tag{2.13}$$

where $t \in G_r^0$, $u \in G_g^0$ and $v \in G_b^0$ denote the support of X-stabilizers in $\mathcal{SC}_r$, $\mathcal{SC}_g$ and $\mathcal{SC}_b$, respectively, and $|\cdot|$ denotes the Hamming weight. We can expand $(X_r + t) \circ (X_g + u) \circ v$ as follows:

$$(X_r + t) \circ (X_g + u) \circ v = (X_r \circ X_g \circ v) + (X_r \circ u \circ v) + (t \circ X_g \circ v) + (t \circ u \circ v). \tag{2.14}$$

By Lemma 1, the parity of $(X_r + t) \circ (X_g + u) \circ v$ is equal to the parities of each term in the expansion shown in Equation 2.14. Let us calculate the parity of each of these terms in turn.

First, consider $(t \circ u \circ v)$. By Lemma 2, the bit-wise product of t and u is equal to the vertex support of a Z-stabilizer of $\mathcal{SC}_b$. As v denotes the support of an X-stabilizer of $\mathcal{SC}_b$ and stabilizers of the same code commute, $|t \circ u \circ v| = 0 \mod 2$ for any t, u and v. Next, consider $(X_r \circ u \circ v)$. By Lemma 2, $u \circ v$ is equal to the vertex support of a Z-stabilizer of $\mathcal{SC}_r$. Logical operators and stabilizers of the same code commute, which implies that $|X_r \circ u \circ v| = 0 \mod 2$ for any u and v. An analogous argument holds for $|t \circ X_b \circ v|$. Finally, $|X_r \circ X_g \circ v| = 0 \mod 2$ for any v, by assumption. We have found that every term in the expansion in Equation 2.14 has even Hamming weight, which implies that the sum has even Hamming weight. Therefore, the overlap of any X-stabilizer of $\mathcal{SC}_b$ with any logical X-operators of $\mathcal{SC}_r$ and $\mathcal{SC}_g$ is even. An identical argument holds for the logical X-operators of any other pair of codes.

Next, we prove the second property. The parity of the overlap of any three logical-X operators is given by the following formula:

$$|(X_r + t) \circ (X_g + u) \circ (X_b + v)| \mod 2, \tag{2.15}$$

where $t \in G_r^0$, $u \in G_g^0$ and $v \in G_b^0$ again denote the support of X-stabilizers in $\mathcal{SC}_r$, $\mathcal{SC}_g$ and $\mathcal{SC}_b$, respectively. We can expand $(X_r + t) \circ (X_g + u) \circ (X_b + v)$ as follows:

$$\begin{aligned}(X_r + t) \circ (X_g + u) \circ (X_b + v) = {} & X_r \circ X_g \circ X_b \\ & + (X_r \circ X_g \circ v) + (X_r \circ u \circ X_b) + (t \circ X_g \circ X_b) \\ & + (X_r \circ u \circ v) + (t \circ X_g \circ v) + (t \circ u \circ X_b) \\ & + t \circ u \circ v.\end{aligned} \tag{2.16}$$

By Lemma 1, the parity of $(X_r + t) \circ (X_g + u) \circ (X_b + v)$ is equal to the sum of the parities of each term in the expansion shown in Equation 2.16. By the argument for the previous property, we know that every term except $(X_r \circ X_g \circ X_b)$ has even Hamming weight, for any t, u, and v. The parity of $(X_r + t) \circ (X_g + u) \circ (X_b + v)$ is therefore determined by the parity of $(X_r \circ X_g \circ X_b)$, which is odd by assumption. Therefore, the overlap of any three logical X-operators (from different codes) is odd. □

We are now ready to state and prove our main theorem.

Theorem 3 *Consider a stack of three 3D surface codes defined on the same rectified-picture lattice, where each code has one encoded qubit. Suppose that that the stack of codes satisfies the following conditions:*

i The overlap of any two X-stabilizer generators of different codes is equal to the vertex support of a Z-stabilizer of the third code.

ii There exist three $\overline{X}$-operators (one from each code) with the following properties:

a The overlap of any two of the $\overline{X}$-operators and the X-stabilizer group of the third code is even.

b The overlap of the three $\overline{X}$-operators is odd.

Then, applying CCZ gates to each triple of physical qubits (one from each code) at the vertices of the lattice implements a logical CCZ gate.

Proof. Define $U = CCZ^{\otimes n}$, where each CCZ gate acts on the three qubits (one per code) at one of the n vertices of the lattice. We consider the initial state

$$|\overline{\alpha\beta\gamma}\rangle_{rgb} = \sum_{\substack{t\in G_r^\alpha \\ u\in G_g^\beta \\ v\in G_b^\gamma}} |t\rangle_r\,|u\rangle_g\,|v\rangle_b\,, \tag{2.17}$$

where $\alpha,\beta,\gamma \in \{0,1\}$. We have omitted the normalization factor. Now, we apply U to $|\overline{\alpha\beta\gamma}\rangle_{rgb}$:

$$\begin{aligned} U|\overline{\alpha\beta\gamma}\rangle_{rgb} &= \sum_{\substack{t\in G_r^\alpha \\ u\in G_g^\beta \\ v\in G_b^\gamma}} CCZ^{\otimes n}\,|t\rangle_r\,|u\rangle_g\,|v\rangle_b\,, \\ &= \sum_{\substack{t\in G_r^\alpha \\ u\in G_g^\beta \\ v\in G_b^\gamma}} (-1)^{|t\circ u\circ v|}\,|t\rangle_r\,|u\rangle_g\,|v\rangle_b\,, \end{aligned} \tag{2.18}$$

where $u\circ v$ denotes the bit-wise (binary) product between u and v and $|t|$ denotes the Hamming weight of t. We want to show that U implements a logical CCZ, which has the following action on the encoded computational basis states:

$$\overline{CCZ}|\overline{\alpha\beta\gamma}\rangle = \begin{cases} -|\overline{\alpha\beta\gamma}\rangle & \alpha=\beta=\gamma=1, \\ \ \ |\overline{\alpha\beta\gamma}\rangle & \text{else}. \end{cases} \tag{2.19}$$

To show that U implements the correct logical operator, we calculate $(-1)^{|t\circ u\circ v|}$ for each encoded computational basis state. We can expand $t\circ u\circ v$ as follows:

$$t\circ u\circ v = (\alpha X_r + t')\circ(\beta X_g + u')\circ(\gamma X_b + v'), \tag{2.20}$$

where $t' \in G_0^r$, $u' \in G_0^g$ and $v' \in G_0^b$. X_r is an n-bit vector describing the support of a $\overline{X}_r$-operator (likewise for X_g and X_b). We emphasize that $(X_r + t')$ describes the support of an operator that is logically equivalent to X_r.

First, consider the exponent for $|\overline{000}\rangle$, which is equal to $|(t'\circ u'\circ v')|$. By

Lemma 2, the overlap any two X-stabilizers of $\mathcal{SC}_r$ and $\mathcal{SC}_g$ is equal to the overlap of a Z-stabilizer of $\mathcal{SC}_b$. As the stabilizers of the same code commute, $|t' \circ u' \circ v'| = 0$ mod 2 for all t', u' and v' and therefore $(-1)^{|t \circ u \circ v|} = 1$ for $|\overline{000}\rangle$. Next, consider the exponent for $|\overline{001}\rangle$, which is equal to $|t' \circ u' \circ (X_b + v')|$. We know that the overlap any two X-stabilizers of $\mathcal{SC}_r$ and $\mathcal{SC}_g$ is equal to the overlap of a Z-stabilizer of $\mathcal{SC}_b$. Logical operators and stabilizers of the same code commute, so the overlap of $\overline{X}_b$ with any pair of stabilizers from the other two codes is always even. This implies that $|t' \circ u' \circ (X_b + v')| = 0 \mod 2$ for every t', u' and v'. Therefore $(-1)^{|t \circ u \circ v|} = 1$ for $|\overline{001}\rangle$. We can make an analogous argument for the states $|\overline{100}\rangle$ and $|\overline{010}\rangle$.

The next computational basis state we consider is $|\overline{110}\rangle$. The exponent for this state is $|(X_r + t') \circ (X_g + u') \circ v'|$. By Lemma 3, the overlap of any $\overline{X}_r$ operator, $\overline{X}_g$ operator and $\mathcal{SC}_b$ stabilizer is even, so $|(X_r + t') \circ (X_g + u') \circ v'| = 0 \mod 2$ for all t', u' and v'. Hence, $(-1)^{|t \circ u \circ v|} = 1$ for $|\overline{110}\rangle$. The same is true for $|\overline{101}\rangle$ and $|\overline{011}\rangle$ by an identical argument. Finally, for the state $|\overline{111}\rangle$ we must consider the exponent $|(X_r + t') \circ (X_g + u') \circ (X_b + v')|$. By Lemma 3, any three logical X-operators (from different codes) have odd overlap so $|(X_r + t') \circ (X_g + u') \circ (X_b + v')| = 1 \mod 2$ for all t', u' and v', which implies that $(-1)^{|t \circ u \circ v|} = -1$ for $|\overline{111}\rangle$. □

Given the proof of Theorem 3, to show that a stack of three 3D surface codes has a transversal $\overline{CCZ}$, we only need to show that the codes satisfy the conditions of the theorem. We note that the conditions in Theorem 3 are very similar to those in the definition of tri-orthogonal codes, which we discussed in Section 1.3.1. The main difference is that the conditions in our theorem concern the overlap of stabilizers and logical operators in different codes whereas tri-orthogonality concerns the overlap of stabilizers and logical operators in the same code.

The family of codes we described in Section 2.1.3 are an example of a family of stacked 3D surface codes with a transversal CCZ. We recall that these codes are defined on rectified cubic lattices with boundaries (see Figure 2.11). Consider the first condition of Theorem 3 (stabilizer overlap). From the definition of the stabilizers (see Table 2.1), it is clear that the overlap of X-stabilizer generators from different codes in the bulk is equal to the vertex support of a Z-stabilizer of the third code. However, this condition is not necessarily satisfied stabilizers on the boundaries, because they can overlap on edges. However, by inspecting Figure 2.11,

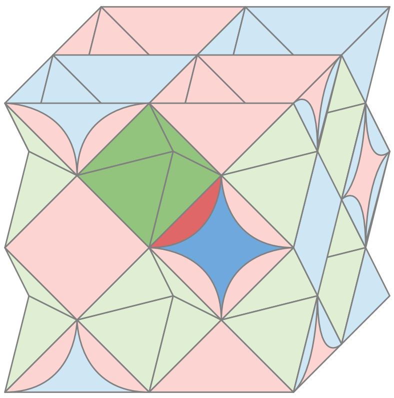


Figure 2.14: The overlap of a stabilizers on the boundaries of a rectified cubic lattice. A $\mathcal{SC}_g$ X-stabilizer (dark green) and a $\mathcal{SC}_r$ X-stabilizer (dark blue) overlap on an edge (dark red). A $\mathcal{SC}_b$ Z-stabilizer is associated with this edge, as explained in Figure 2.11. Figure adapted from Vasmer and Browne [112].

we conclude that the overlap of any pair of X-stabilizers (from different codes) that are associated with lattice elements on the boundaries is equal to the vertex support of a Z-stabilizer of the third code. We illustrate one of the cases where stabilizers overlap on an edge in Figure 2.14.

Next, we consider the second condition of Theorem 3 (logical operator overlap). By inspecting Figure 2.13, we see that the overlap of any two canonical $\overline{X}$-operators of different codes is equal to the vertex support of the canonical $\overline{Z}$-operator of the third code. This implies that any two canonical $\overline{X}$-operators of different codes have even overlap with the X-stabilizers of the third code, because logical operators and stabilizers of the same code commute. In addition, the fact that the overlap of any two canonical $\overline{X}$-operators of different codes is equal to the vertex support of the canonical $\overline{Z}$-operator of the third code implies that the three canonical $\overline{X}$-operators have odd overlap, because logical X- and Z-operators acting on the same qubit anticommute. We have shown that our family of 3D surface codes with boundaries satisfy the assumptions of Theorem 3. Therefore, these codes have a transversal CCZ gate.

2.2.1 Transversal CZ

Stacks of 3D surface codes that have a transversal CCZ also have a transversal CZ. We make this statement precise in the following corollary.

Corollary 1 *Consider a stack of three 3D surface codes defined on the same rectified-picture lattice, where each code has one encoded qubit. Let $\overline{X}_c$ denote a logical X-operator of the code $\mathcal{SC}_c$, where $c \in \{r, g, b\}$. Assume that $CCZ^{\otimes n}$ implements $\overline{CCZ}$, where the physical CCZ gates act on the triple of qubits at each of the n vertices of the lattice. Then, applying CZ gates to the physical qubits of $\mathcal{SC}_c$ and $\mathcal{SC}_{c'}$ at the vertices in the vertex support of a $\overline{X}_{c''}$-operator implements $\overline{CZ}_{cc'}$, where $\overline{CZ}_{cc'}$ denotes a logical CZ operator acting on the encoded qubits of $\mathcal{SC}_c$ and $\mathcal{SC}_{c'}$.*

Proof. We recall the definition of the group commutator of two operators A and B: $K[A,B] = ABA^{\dagger}B^{\dagger}$. One can easily verify that $K[CCZ, X_k] = CZ_{ij}$, where i, j and k label three qubits and the operator subscripts denote which qubits the respective operators act on (CCZ acts on all three so we omit the indices for this operator). Therefore, we can implement $\overline{CZ}_{rg}$ by applying the product of transversal logical operators $K[\overline{CCZ}, \overline{X}_b]$. Now, consider the action of these transversal logical operators on the physical qubits of the three codes. All triples of qubits at vertices that are not in the vertex support of $\overline{X}_b$ are acted upon by $CCZ \times CCZ^{\dagger} = I$. All triples of qubits at vertices that are in the vertex support of $\overline{X}_b$ are acted upon by $K[CCZ, X_b] = CZ_{rg}$. Therefore, we can implement a transversal $\overline{CZ}_{rg}$-operator by applying CZ gates to pairs of physical qubits (one from $\mathcal{SC}_r$ and one from $\mathcal{SC}_g$) that lie on the vertices in the vertex support of a $\overline{X}_b$-operator. The argument for $\overline{CZ}_{rb}$ and $\overline{CZ}_{gb}$ is identical. □

2.2.2 Universal gate set

We have shown that stacks of 3D surface codes have transversal CCZ and CZ gates. However, these two gates do not form a universal set of gates. For this, we also need another gate, e.g. the Hadamard gate [61]. However, this gate cannot be transversal without violating the no-go result of Eastin and Knill [62], which states that any code that can correct any single-qubit error cannot have a universal and transversal set of encoded gates. Therefore, we need to find a method for implementing a Hadamard in

3D surface codes. Fortunately, this can be achieved using CZ, $|+\rangle$ state preparation and an X-basis measurement, as shown in Figure 2.15. We have just shown that CZ is transversal in 3D surface codes and we can accomplish the other two operations using state preparation and measurement procedures that are completely analogous to those used in 2D surface codes (which we described in Section 1.3.1). Using the circuit in Figure 2.15 and our transversal CCZ gate, we can implement a fault-tolerant and universal set of gates using 3D surface codes with smaller time overhead than is required in 2D surface codes. Whether the overall overhead is smaller is a more subtle question, which we return to in Chapter 4.

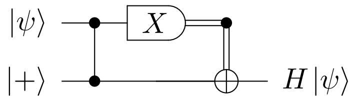


Figure 2.15: A circuit which implements a H-gate using $|+\rangle$ state preparation, X-basis measurement and CZ [78].

2.3 Generalizations of the rectified picture

We discovered the rectified picture by generalizing the rotated picture of 2D surface codes. Therefore, it is natural to ask whether we can generalize the rectified picture. From a more practical standpoint, our family of 3D surface codes (see Section 2.1.3) may not be the most efficient in terms of number of logical qubits and code distance. Generalizing the rectified picture may enable us to find more efficient codes with a transversal CCZ. In this section, we first try to find other examples of rectified-picture lattices in 3D. Then we consider generalizations of the rectified picture to higher dimensions and curved spaces.

Our first task is find other lattices that support three 3D surface codes in the rectified picture. We say a lattice supports n lattices in the rectified picture if we can partition the cells and faces into n sets in such a way that we can define a valid 3D surface code for each set. In 2D there are many lattices that support two surface codes in the rotated picture. For example, the lattices in Figure 2.2 support two different 2D surface codes (either we have X-stabilizers on red faces and Z-stabilizers on blue faces or we have X-stabilizers on blue faces and Z-stabilizers on red faces). Indeed, any planar 4-valent lattice will support two surface codes in the rotated picture [115, Section 5].

In 3D, we have not found any other any convex uniform lattices that support three 3D surface codes in the rectified picture. A convex uniform 3D lattice is a tiling of 3D space by non-overlapping uniform polyhedra, where uniform means that every polyhedron (and the lattice) are vertex-transitive [113, 120]. We have found an example of a convex uniform lattice that supports four 3D surface codes in the rectified picture: the cubic lattice. Suppose that we colour the cells of the cubic lattice with four colours so that each cube has the same colour as the cubes with which it shares exactly one vertex (see Figure 2.16). With this colouring the cubic lattice supports four 3D surface codes in the rectified picture. We place four qubits at each vertex (one per code) and we index the four codes with the colours $c \in \{r, g, b, y\}$. The stabilizer generators of the four codes are shown in Table 2.2. The

Code	X-stabilizers	Z-stabilizers
$\mathcal{SC}_r$	r-cells	bg-faces, by-faces and gy-faces
$\mathcal{SC}_g$	g-cells	rb-faces, ry-faces and by-faces
$\mathcal{SC}_b$	b-cells	rg-faces, ry-faces and gy-faces
$\mathcal{SC}_y$	y-cells	rb-faces, rg-faces and bg-faces

Table 2.2: The stabilizer generators of the four 3D surface codes supported on the same cubic lattice (in the rectified picture).

idea of defining a 3D surface code the cubic lattice in this way is due to Kubica [121]. However, he did not consider multiple 3D surface codes defined on the same cubic lattice.

Remarkably, the cubic lattice surface codes we have just defined are gauge choices of the 3D Bacon-Shor code [122], a well-known subsystem code. We recall that in a subsystem code, the encoded qubits can be divided into logical qubits (which we use to store information) and gauge qubits (whose state we don't care about). To fix the gauge of a subsystem code, we fix the gauge qubits to be in some stabilizer state (a stabilizer state is a $k = 0$ stabilizer code). It is widely-known that the 2D surface code is a gauge choice of the 2D Bacon-Shor code, see e.g. [118, Section 2].

In the 3D Bacon-Shor code, we place qubits on the vertices of a cubic lattice. We use a coordinate system where the edges of the cubic lattice are parallel to either $\hat{\mathbf{\imath}}$, $\hat{\mathbf{\jmath}}$ or $\hat{\mathbf{k}}$ (the three unit vectors in the standard basis). The gauge group of the 3D Bacon-Shor code is generated by $X \otimes X$ and $Z \otimes Z$ operators associated with the

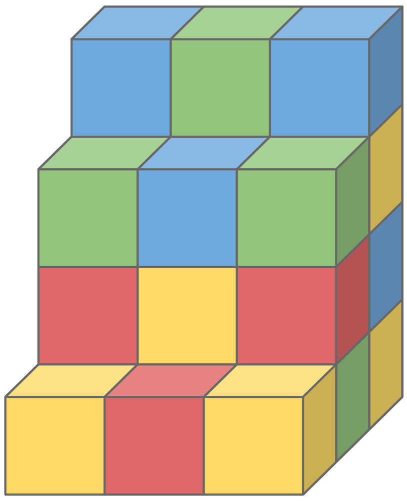


Figure 2.16: A cubic lattice coloured with four colours. Cubes which share exactly one vertex have the same colour. Figure adapted from Vasmer and Browne [112].

edges of the cubic lattice. The X gauge generators are associated with edges parallel to $\hat{\mathbf{i}}$ and $\hat{\mathbf{j}}$. The Z gauge generators are associated with edges parallel to $\hat{\mathbf{j}}$ and $\hat{\mathbf{k}}$. The stabilizer group of the 3D Bacon-Shor code is generated by "nearest plane" operators. That is, the X-stabilizer generators consist of X-operators acting on all the qubits in two $\hat{\mathbf{j}}$-$\hat{\mathbf{k}}$ planes (planes parallel to $\hat{\mathbf{j}}$ and $\hat{\mathbf{k}}$) that are adjacent in the $\hat{\mathbf{i}}$ direction. Analogously, the Z-stabilizer generators consist of Z-operators acting on all the qubits in two $\hat{\mathbf{i}}$-$\hat{\mathbf{j}}$ planes that are adjacent in the $\hat{\mathbf{k}}$ direction.

A stabilizer code defined by a stabilizer group $\mathcal{S}$ is a gauge choice of a subsystem code defined by the gauge group $\mathcal{G}_1$ if the following inclusions hold [72, 97]:

$$\mathcal{S}_1 \subseteq \mathcal{S} \subseteq \mathcal{G}_1, \tag{2.21}$$

where $\mathcal{S}_1 = \mathcal{Z}(\mathcal{G}_1)$ is the stabilizer group of the subsystem code. Consider $\mathcal{SC}_r$ as defined in Table 2.2. The X-stabilizer generators of $\mathcal{SC}_r$ are associated with r-cubes. We can construct these operators from the X gauge generators of the 3D Bacon-Shor code ($X \otimes X$ operators associated with edges parallel to either $\hat{\mathbf{i}}$ or $\hat{\mathbf{j}}$). The Z-stabilizer generators of $\mathcal{S}_r$ are associated with bg-, by- and gy-faces. We can construct these operators from the Z gauge generators of the 3D Bacon-Shor code ($Z \otimes Z$ operators associated with edges parallel to either $\hat{\mathbf{j}}$ or $\hat{\mathbf{k}}$). Therefore, the gauge group of the 3D Bacon-Shor code contains the stabilizer group of $\mathcal{SC}_r$.

Now consider the stabilizers of the 3D Bacon-Shor code. We can construct any X $\hat{\mathbf{j}}$-$\hat{\mathbf{k}}$ nearest plane operator from the X-stabilizer generators of $\mathcal{SC}_r$ (r-cubes). We can construct any $\hat{\mathbf{i}}$-$\hat{\mathbf{j}}$ Z nearest plane operators from the Z-stabilizer generators of $\mathcal{SC}_r$ (bg-, by- and gy-faces). This can seen by considering the relevant cells and faces of the cubic lattice in Figure 2.16. Therefore, the stabilizer group of $\mathcal{SC}_r$ contains the stabilizer group of the 3D Bacon-Shor code, which implies that $\mathcal{SC}_r$ is a gauge choice of the 3D Bacon-Shor code. The same argument holds for all the other 3D surface codes defined in Table 2.2 by symmetry.

We have managed to find one other example of a rectified-picture lattice in 3D, the cubic lattice. Using this lattice may have advantages over the rectified cubic lattice, due to its regularity and the fact that the surface codes defined on it have maximum stabilizer weight of eight. Also, the fact that these surface codes are gauge choices of the 3D Bacon-Shor code means that we should be able to measure gauge operators instead of stabilizers, reducing the weight of the operators we have to measure still further. However, to fully understand whether the cubic lattice is preferable, we need to check if CCZ is transversal for codes defined on this lattice (with and without boundaries). We also need to investigate the decoding problem and estimate the error threshold. We leave these tasks for future work.

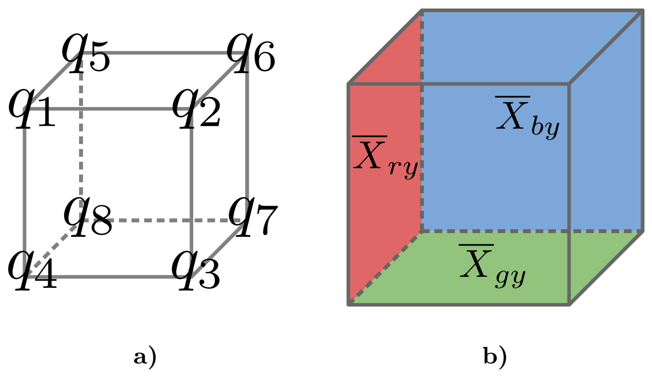
2.3.1 Transforming surface codes into colour codes

In the next section, we present a generalization of the rectified picture to higher dimensions and to hyperbolic space. But before we do this, it is useful to consider the relationship between the rectified picture and colour codes in more detail. We illustrate this connection with an example, showing that one can transform three 3D surface codes into a colour code by encoding the physical qubits of the surface codes using small colour codes. This provides a simple implementation of the local Clifford circuits for transforming surface codes into colour codes which have been proven to exist [39]. Our implementation is a generalization of a result due to Criger and Terhal who showed that one can transform a pair of 2D surface codes into a 2D colour code by encoding pairs of physical qubits in the [[4,2,2]] error detecting code [123]. In our 3D construction, we use the [[8,3,2]] code instead of the [[4,2,2]] code.

The [[8,3,2]] code can be regarded as a 3D colour code defined on a cube with qubits on the vertices (see Figure 2.17). We recall that in 3D colour codes, we place qubits at vertices, we associate X-stabilizer generators with cells, and we associate Z-stabilizer generators with faces. Therefore, the X-stabilizer of the [[8,3,2]] code acts on all the qubits and the Z-stabilizer generators are associated with the faces of the cube. The $\overline{X}$-operators are products of X-operators acting on all qubits on the same face (opposite faces support $\overline{X}$-operators that act on the same encoded qubit). The $\overline{Z}$-operators are products of Z-operators acting on the qubits at the endpoints of edges linking the faces that support the corresponding $\overline{X}$-operators. The 1-skeleton (vertices and edges) of a cube is a bipartite graph, and we use this to divide the physical qubits into two sets. We can implement a transversal logical $\overline{CCZ}$-gate in the [[8,3,2]] code by applying T and $T^\dagger$ respectively to the qubits in each of the bipartite sets. This fact can be verified by computing the action of T and $T^\dagger$ on the codeword kets [124].

In a 3D colour code, we can assign faces the colours of the cells they are members of. For example, a face that is a member of a c-cell and a c'-cell is a cc'-face. Due to the 4-colourability of the cells in a 3D colour code lattice, each cell's faces are 3-colourable. Consider a 3D colour code lattice where each cell is assigned a colour from the set $\{r, g, b, y\}$. The [[8,3,2]] code could be the cell of such a lattice, say with colour y. Then the its faces would be coloured ry, gy and by. We use these colours to index the logical operators of the [[8,3,2]] code. That is, we denote the $\overline{X}$-operators that are supported on the cy-faces by $\overline{X}_{cy}$, where $c \in \{r, g, b\}$. Analogously, we use $\overline{Z}_{cy}$ to denote the $\overline{Z}$-operators that are supported on edges that link the cy-faces. This notation is illustrated in Figure 2.17.

Now that we have introduced the [[8,3,2]] code, we can describe the transformation that maps three 3D surface codes to a single 3D colour code. Consider three 3D surface codes defined on a rectified cubic lattice that tessellates the 3-torus. We assume that the cells of the rectified cubic lattice have the same colours as in Section 2.1.2. To transform the three codes, we encode the three physical qubits at each vertex of lattice using the [[8,3,2]] code. More specifically, we encode the physical qubits of $\mathcal{SC}_c$ as the cy-qubits of the [[8,3,2]] codes. This allows us to understand the transformation geometrically. For example, consider a $\mathcal{SC}_g$ X-stabilizer



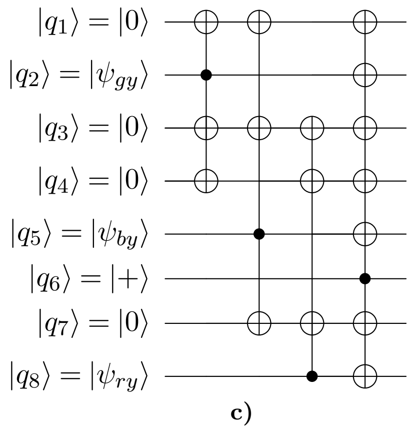

$|q_1\rangle = |0\rangle$
$|q_2\rangle = |\psi_{gy}\rangle$
$|q_3\rangle = |0\rangle$
$|q_4\rangle = |0\rangle$
$|q_5\rangle = |\psi_{by}\rangle$
$|q_6\rangle = |+\rangle$
$|q_7\rangle = |0\rangle$
$|q_8\rangle = |\psi_{ry}\rangle$
c)

Figure 2.17: The [[8,3,2]] code as a 3D colour code. In Figure 2.17a, we show our labelling of the qubits, which we use in the encoding circuit shown in Figure 2.17c. In Figure 2.17b, we show the [[8,3,2]] code as if it was a cell in a larger 3D colour code. The highlighted $\overline{X}_{cy}$-operators act on the encoded qubits corresponding to $|\psi_{cy}\rangle$ in Figure 2.17c. Figure adapted from Vasmer and Browne [112].

associated with a g-octahedron. Each X-operator acting on the individual physical qubits is mapped to an $\overline{X}_{gy}$-operator by the [[8,3,2]] encoding circuit. Therefore, the weight six $\mathcal{SC}_g$ stabilizer is mapped to a weight twenty-four stabilizer. Geometrically, this corresponds to truncating the octahedron so that it becomes a truncated octahedron. All other stabilizers transform in a similar way: cuboctahedra become truncated cuboctahedra, squares become octagons, triangles become hexagons and vertices become cubes.

The transformed stabilizers commute with each other because the encoding circuit of the [[8,3,2]] code preserves the commutation of Pauli operators acting on the input physical qubits ($|\psi_{cy}\rangle$ in Figure 2.17c). The logical operators of the 3D surface codes are mapped to logical operators in the 3D colour code for the same

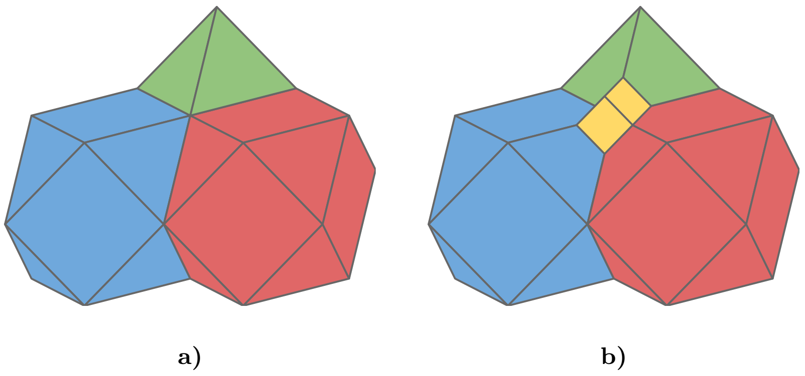


Figure 2.18: Applying the [[8,3,2]] transformation to a single vertex of a rectified cubic lattice. In Figure 2.18a, we show part of the initial rectified cubic lattice. In Figure 2.18b, we have encoded the three qubits at the vertex where the cells meet in an [[8,3,2]] code. Geometrically, this corresponds to replacing the vertex with a cube (yellow cell). Figure adapted from Vasmer and Browne [112].

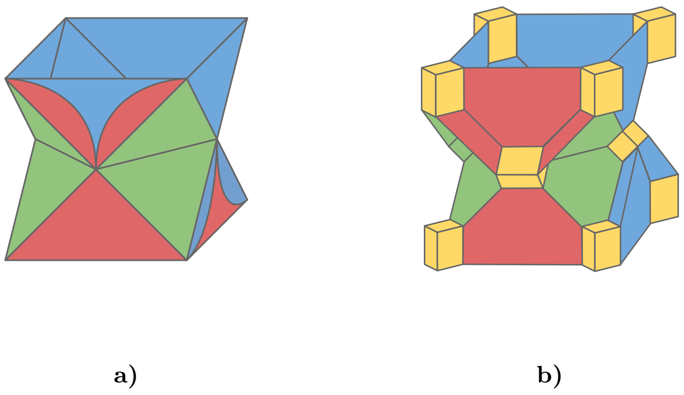


Figure 2.19: Transforming a stack of three 3D surface codes into a single 3D colour code. The lattice in Figure 2.19a supports three distance two 3D surface codes. It is the smallest non-trivial member of the family of 3D surface codes with boundaries defined in Section 2.1.3. Each vertex in Figure 2.19a is transformed as shown in Figure 2.18. The resultant lattice (Figure 2.19b) is a distance four 3D colour code. Figure adapted from Vasmer and Browne [112].

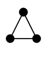
reason. Globally, the rectified cubic lattice is transformed into a cantitruncated cubic lattice (one of the vertex-regular lattices that supports a 3D colour code). In a cantitruncated cubic lattice, two truncated cuboctahedra, one truncated octahedron and one cube meet at each vertex. Figure 2.18 shows how a single vertex transforms and Figure 2.19 shows how a small stack of surface codes transforms.

2.3.2 Codes from Coxeter diagrams

In this Section, we generalize the rectified picture to higher dimensions and to curved spaces. Because we can no longer rely on visualizations, we need more sophisticated mathematical tools. We restrict our attention to uniform (vertex-transitive) tessellations because these tessellations have a compact representation in terms of such a mathematical tool: Coxeter groups [113, Chapter 11]. We specify Coxeter groups using a group presentation $< S|R >$, where S is a generating set of the group and R are relations amongst the generators. Using this language, Coxeter groups are groups with the following presentation:

$$\langle R_1, R_2, \dots R_m | (R_i R_j)^{p_{ij}} = 1 \rangle, \tag{2.22}$$

where $p_{ii} = 1$ and $p_{ij} \geq 2$. If $p_{ij} = \infty$, then the corresponding relation is ignored. The R_i can be thought of as reflections. We restrict ourselves to Coxeter groups whose fundamental regions are simplices, where for a given group the R_i correspond to the facets of the fundamental region simplex. A D-dimensional simplex is the generalization of the triangle to D-dimensions. For example, a 0D simplex is a point, a 1D simplex is a line, a 3D simplex is a tetrahedron, and a 4D simplex is a tetrahedral pyramid. The fundamental region of a reflection group is a region whose reflections cover the whole space without overlapping (and without gaps). We make the restriction to simplicial fundamental regions because it is unclear how the rectified picture should be defined for non-simplicial fundamental regions.

Coxeter groups are nicely represented by graphs, called Coxeter diagrams. In a given Coxeter diagram, the vertices represent the facets of the fundamental region, where a facet is a $(D-1)$-cell of a D-dimensional simplex. Edges represent the dihedral angles between different facets. Unmarked edges correspond to dihedral angles of $\pi/3$ whereas edges marked by $p > 3$ correspond to dihedral angles of π/p. There are no edges between vertices that correspond to perpendicular facets. For example, $\bullet\overset{4}{-}\bullet\overset{4}{-}\bullet$ represents the Coxeter group whose fundamental region is a right-angled isosceles triangle and  represents the Coxeter group whose fundamental region is an equilateral triangle.

We use Wythoff's (kaleidoscopic) construction [113, 125] to construct tessellations from Coxeter diagrams. Given a Coxeter diagram, we mark a subset of its

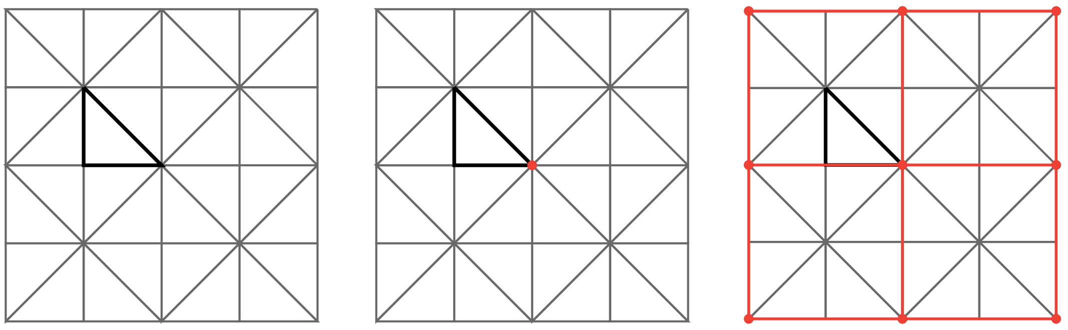


Figure 2.20: Wythoff's construction applied to $\bullet\overset{4}{-}\bullet\overset{4}{-}\bullet$. In Figure 2.20a, we highlight the fundamental region in black. In Figure 2.20b, we show the generating point of the tessellation corresponding to $\circ\overset{4}{-}\bullet\overset{4}{-}\bullet$ in red, and in Figure 2.20c, we show the full tessellation (square lattice).

vertices. Then we imagine a generating point inside the fundamental domain, where the generating point is equidistant from all the facets that correspond to marked vertices. If a vertex is unmarked, then the generating point lies on the corresponding facet. The tessellation is the reflection of the generating point in all of the facets. The easiest way to illustrate Wythoff's construction is via an example, which we provide in Figure 2.20.

The useful thing about marked Coxeter diagrams is that different mark-ups of a Coxeter diagram correspond to surface code lattices in different pictures and their related colour codes. For example, $\circ\overset{4}{-}\bullet\overset{4}{-}\bullet$ corresponds to the square lattice (see Figure 2.20). If we define a 2D surface code on this lattice, then $\bullet\overset{4}{-}\bullet\overset{4}{-}\circ$ corresponds to the dual lattice of this code and $\bullet\overset{4}{-}\circ\overset{4}{-}\bullet$ corresponds to the rotated picture lattice of the same 2D surface code. Furthermore, $\circ\overset{4}{-}\circ\overset{4}{-}\circ$ corresponds to the 2D colour code that is local-Clifford equivalent to a pair of surface codes defined on $\circ\overset{4}{-}\bullet\overset{4}{-}\bullet$ and $\bullet\overset{4}{-}\bullet\overset{4}{-}\circ$. Figure 2.21 shows these lattices. The fact that colour code lattices correspond to Coxeter diagrams with all vertices marked has previously been noticed [126]. Furthermore, this construction is similar to a method for constructing colour code lattices presented in [95, Appendix A].

Thus far we have considered 2D tessellations, but we can also consider higher dimensions. For example, the Coxeter diagram, $\begin{smallmatrix}\bullet & & \\ & \bullet & \overset{4}{-}\bullet \\ \bullet & & \end{smallmatrix}$, corresponds to a reflection group whose fundamental domain is a certain trirectangular tetrahedron (shown in Figure 2.22), i.e. a tetrahedron with three faces that meet at right angles. The

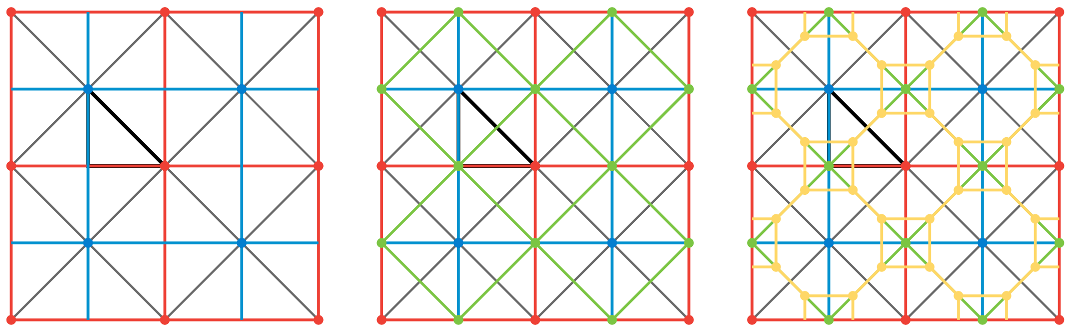


Figure 2.21: Related topological code lattices with the same symmetry. In Figure 2.21a, we show the lattices corresponding to the Coxeter diagrams $\circ\overset{4}{-}\bullet\overset{4}{-}\bullet$ (red) and $\bullet\overset{4}{-}\bullet\overset{4}{-}\circ$ (blue). In Figure 2.21b, we add the lattice corresponding to $\bullet\overset{4}{-}\circ\overset{4}{-}\bullet$ (green). Finally, in Figure 2.21c, we add the lattice corresponding to $\circ\overset{4}{-}\circ\overset{4}{-}\circ$ (yellow).

marked Coxeter diagram, ${}^{\bullet}_{\bullet}\!\!>\!\bullet\overset{4}{-}\circ$, corresponds to the cubic lattice; ${}^{\circ}_{\bullet}\!\!>\!\bullet\overset{4}{-}\bullet$ and ${}^{\bullet}_{\circ}\!\!>\!\bullet\overset{4}{-}\bullet$ correspond to tetrahedral-octahedral lattices; ${}^{\bullet}_{\bullet}\!\!>\!\circ\overset{4}{-}\bullet$ corresponds to the rectified cubic lattice; and ${}^{\circ}_{\circ}\!\!>\!\circ\overset{4}{-}\circ$ corresponds to the cantitruncated cubic lattice. These are exactly the 3D surface code lattices and related 3D colour code lattices that we examined earlier in this chapter.

We can spot a pattern developing as we go from two to three dimensions. If we want to define multiple surface codes on the same lattice in the rectified picture, we need to find Kitaev picture lattices that can be arranged so that every edge of each lattice intersects an edge from each of the other lattices (with exactly one such intersection point for each edge). When this is the case, we can place rectified-picture vertices at the points where the edges of the Kitaev picture lattices intersect. In 2D, this is possible for any tessellation described by a Coxeter diagram whose fundamental region is a right-angled triangle ($\bullet\overset{4}{-}\bullet\overset{4}{-}\bullet$ and $\bullet\!-\!\bullet\overset{6}{-}\bullet$ [113, pp. 297]). And in 3D, we can construct rectified-picture lattices using any Coxeter diagram whose fundamental region is some trirectangular tetrahedron. For such a fundamental region, the three edges of the tetrahedron correspond to the edges of the (Kitaev picture) surface code lattices, as we show in Figure 2.22. As every point in the tessellation is just some reflection of the fundamental region, the edges of the three lattices intersect in just the way we want. The only Coxeter diagram with a

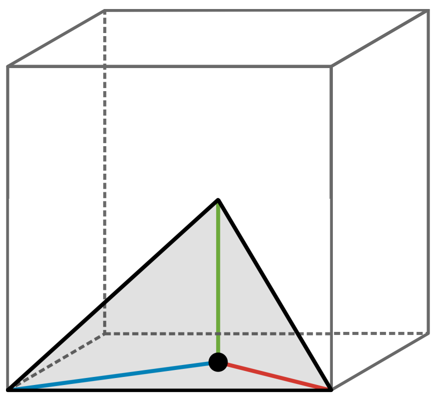
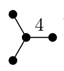
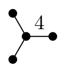
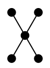


Figure 2.22: The (tetrahedral) fundamental region of the Coxeter diagram, , shown inside a cube. We have highlighted the edges (red, green and blue) of the three tessellations formed by marking each of the three outer vertices of the diagram using Wythoff's construction. The point where these edges meet (black circle) is the vertex of the corresponding rectified-picture lattice.

trirectangular tetrahedron fundamental region in 3D Euclidean space is  [113, pp. 297].

In general, to find uniform tessellations that support D, D-dimensional surface codes in D-dimensions, we need to find Coxeter diagrams that look like star graphs with $D+1$ vertices. A star graph on $D+1$ vertices is simply a graph with one central vertex connected to all the other (outer) vertices. The fundamental regions of such a Coxeter diagram is a right-angled simplex (the generalization of a right-angled triangle to D-dimensions). Given such a Coxeter diagram, we construct D Kitaev picture surface codes by marking each of the D outer vertices of the diagram in turn. In each of the resultant tessellations, we place qubits on the edges, we associate X-stabilizer generators with the vertices, and we associate Z-stabilizer generators with the faces. We construct the rectified-picture lattice that simultaneously supports these D surface codes by marking the central vertex of the diagram. The resultant lattice can be obtained via rectification from any of the Kitaev picture lattices. In the rectified-picture lattice, we place D qubits at each vertex, we associate X-stabilizer generators with the 4D cells and we associate Z-stabilizer generators with the faces. There is a one-to-one mapping between the 4D cells of the rectified-picture lattice and the vertices of the Kitaev picture lattices; and there is a one-to-one mapping between the faces of the rectified-picture lattice and the faces of the Kitaev picture lattices. These mapping allow us to work out which rectified-picture stabilizer belongs to which code.

We have already listed all the examples of star-graph Coxeter diagrams in 2D and 3D. In 4D, there is one star-graph Coxeter diagram:  . Table 2.3 lists the tessellations we can construct by marking up this diagram. From this table, we

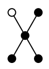
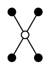
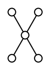
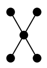
Coxeter Diagram	Lattice
	16-cell lattice
	24-cell lattice
	Truncated 24-cell lattice

Table 2.3: The 4D lattices that can be constructed by applying Wythoff's construction to the Coxeter diagram  .

see that one can arrange four 4D surface codes defined on 16-cell lattices so that the rectification of each lattice is an identical 24-cell lattice. The 16-cell and the 24-cell (also known as the hyper-diamond) are two of the 4D analogues of Platonic solids. Both of these 4D polytopes tessellate 4D Euclidean space [113, Chapter 7]. Furthermore, we conjecture that one can transform the four 4D surface codes into a single 4D colour code by encoding the qubits at rectified-picture vertices in the [[16,4,2]] code (the 4D analogue of the [[8,3,2]] code). We anticipate that the resulting colour code will be defined on uniform 4D lattice where four truncated 24-cells and one tesseract meet at each vertex. The tesseract is the 4D analogue of the cube (see Figure 2.23).

Coxeter diagrams can also be used to describe tessellations of curved spaces e.g. hyperbolic space. Surface codes defined on tessellations of hyperbolic space have previously been studied in [127, 128, 129, 130, 131]. In any dimension $D \geq 2$, the rate of a hyperbolic surface code is constant, i.e. the number of encoded qubits is proportional to the number of physical qubits. This is in contrast to surface codes defined on tessellations of Euclidean space, where the number of encoded qubits is a (small) constant that is independent of the number of physical qubits. In 2D, 3D and 4D, hyperbolic Coxeter groups exist whose Coxeter diagrams are star graphs [132, pp. 141–144], so we can construct rectified-picture lattices in hyperbolic space that

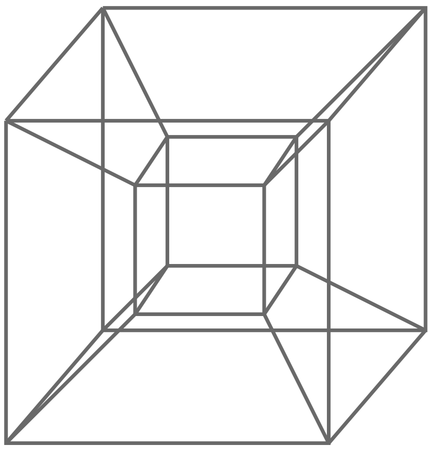


Figure 2.23: A tesseract projected into 3D space.

support multiple surface codes. In both Euclidean and hyperbolic space, there are no star-graph Coxeter diagrams in any dimension $D \geq 5$, we cannot use our method to construct uniform tessellations that support D, D-dimensional surface codes when $D \geq 5$.

Conclusion

In this Chapter, we have shown that 3D surface codes are uncommonly useful for processing quantum information. More specifically, we proved that the non-Clifford CCZ gate is transversal for certain 3D surface codes. This is important because implementing a non-Clifford gate in a fault-tolerant architecture often has high resource requirements. Architectures incorporating 3D surface codes may therefore have smaller overhead than other leading architectures based on topological codes. In Chapter 4, we propose two fault-tolerant quantum computing architectures that incorporate 3D surface codes, and we compare the overhead of our architectures with the overhead of other leading architectures. However, before we can make this comparison, we need to understand how good 3D surface codes are at protecting quantum information. This is the subject of the next chapter.

To prove our transversality result, we introduced the rectified picture of 3D surface codes. Using this picture makes it easier to understand how the stabilizers of different 3D surface codes overlap. In Section 2.3, we considered some generalizations of the rectified picture. In particular, we used Coxeter diagrams to generalize our picture to lattices that exist in four dimensions and/or hyperbolic spaces. Such lattices (in $D \geq 3$ dimensions) may support surface codes with transversal non-

Clifford gates. Hyperbolic surface codes with this property would be especially interesting, because their non-zero rate means that the non-Clifford gate would act on many encoded qubits at the same time. This could be useful for information processing and/or for magic state distillation. We leave further investigation of this possibility to future work.

Chapter 3

Decoding three-dimensional surface codes

In this chapter, we tackle the problem of decoding 3D surface codes. By decoding, we mean using a classical algorithm to process stabilizer measurement outcomes in order to find a correction operator. If the product of the error that occurred and the correction operator is a stabilizer, then we have decoded successfully. As 3D surface codes are CSS codes, we can correct X-errors and Z-errors independently. In Section 3.1, we use a well-established decoder to estimate the Z-error threshold of 3D surface codes. And in Section 3.2, we develop a cellular automaton decoder for X-errors in 3D surface codes with boundaries. We prove that this decoder has an error threshold when stabilizer measurements are perfect, and we numerically estimate the value of the error threshold when stabilizer measurements are unreliable. The results of this chapter are useful for understanding the feasibility of the fault-tolerant quantum computing architectures we present in Chapter 4. The work on the cellular automaton decoder described in this Chapter was carried out in collaboration with Aleksander Kubica and Dan Browne.

The optimal solution to the decoding problem for a stabilizer code returns a correction that maximizes the probability that the product of the error and the correction is a stabilizer. Finding such a correction is called maximum likelihood decoding, and unfortunately no efficient solution to this problem for has been found for 3D surface codes. With an efficient maximum likelihood decoder we would be to find the optimal error threshold of a code (for a given noise model). However, even if we don't have such a decoder, we can still estimate the optimal error thresholds of topological codes. This is because solving the maximum likelihood decoding problem for a topological code can be reduced to finding the partition function of a classical statistical mechanics model [28, 133, 134, 135]. Using such a reduction, the optimal phase-flip error threshold of the 3D surface code has been estimated to be $p_{th}^{Z} \approx 3.3\%$ [136] and the optimal bit-flip error threshold of the 3D surface code has been estimated to be $p_{th}^{X} \approx 23.2\%$ [137, 138]. Both these values are for 3D surface codes defined on the cubic lattice.

3.1 Z-error decoding

The Z-error decoding problem for 3D surface codes is essentially the same as the decoding problem for 2D surface codes. We recall that, in both cases, qubits are on edges and X-stabilizers are associated with vertices (see Section 1.2.1). Therefore, chains of errors on edges create unsatisfied stabilizers at their endpoints. The task of a decoder is to find a collection of edges (a correction) whose endpoints are the unsatisfied stabilizers, so that the product of the correction and the error is a stabilizer. Let us introduce some terminology to state this problem more clearly. We denote the set containing all the i-cells of a lattice $\mathcal{L}$ by $\Delta_i(\mathcal{L})$, e.g. $\Delta_0(\mathcal{L})$ are the vertices of $\mathcal{L}$ and $\Delta_1(\mathcal{L})$ are the edges of $\mathcal{L}$. We define C_i to be an $\mathbb{F}_2$-vector space with a basis formed by the i-cells of the lattice, $\iota \in \Delta_i(\mathcal{L})$. With this definition, we can introduce boundary operators $\partial_i : C_i \to C_{i-1}$, which are linear maps defined for all basis elements $\iota \in \Delta_i(\mathcal{L})$:

$$\partial_i \iota = \sum_{\nu \in \Delta_{i-1}(\iota)} \nu, \tag{3.1}$$

where $\Delta_{i-1}(\iota)$ are the $(i-1)$-cells contained in ι. We emphasize that any vector in C_i is isomorphic to a subset of the i-cells of the lattice.

Using the terminology defined above, Z-errors in 3D surface codes are subsets $\epsilon \subseteq \Delta_1(\mathcal{L})$. The syndrome, σ, corresponding to an error ϵ is the image of ϵ under the 1-boundary map, i.e. $\sigma = \partial_1 \epsilon$. A correction, ζ, is a subset of edges with the same boundary as the error. We know of no efficient algorithm for finding the optimal correction, however we can find a good approximate solution by choosing the correction with the lowest weight. This is equivalent to a weighted graph matching problem, as was first pointed out by Dennis et al. [28]. Given a graph $G = (V, E)$, where V is the node-set of the graph and E is the link-set of the graph, a matching is a set of links without common nodes. We refer to vertices and edges of such a graph as nodes and links to avoid confusion with the elements of the lattice. In a perfect matching, every node of the graph appears in one of the links of the matching. If the links of the graph are weighted, we can also ask to find a matching with minimum weight. Edmonds' Blossom algorithm [31] can be used to solve the minimum-weight matching problem efficiently [32]. We note that in the quantum error-correction research literature, the weighted-graph generalization of the Blossom algorithm is usually called minimum-weight perfect matching (MWPM).

Given a syndrome, we construct its corresponding matching graph, G, as follows. First, we create a node for each unsatisfied stabilizer generator. Then, for each node i, we create a link (i,j) to every other node j with weight equal to the path-length between the corresponding unsatisfied stabilizer generators in the syndrome graph. We note that we can describe such a path using a vector in C_1, which corresponds to the edges in the path. If the code has boundaries, we also have to create nodes for the boundaries [139]. To solve the decoding problem, we need to find a minimum-weight matching in G. Given such a matching, we can construct the correction by taking the sum of the vectors in C_1 that correspond to the links in the matching.

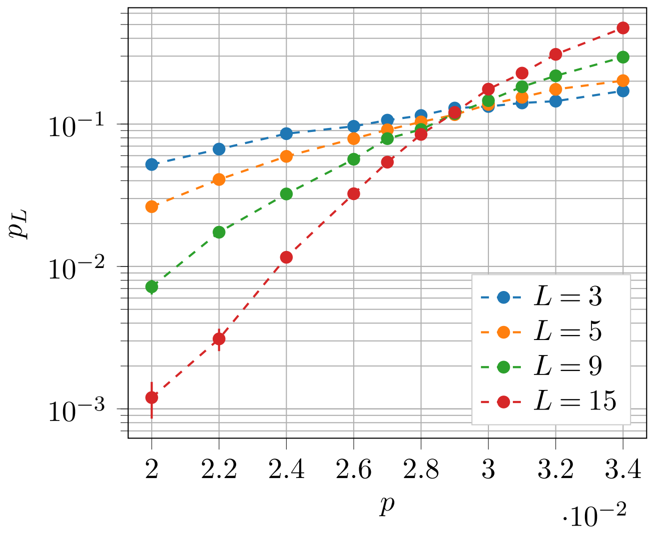
As we discussed in Section 1.2.1, when stabilizer measurements are unreliable the Z-error decoding problem for surface codes is essentially the same as when measurements are perfect [28]. To deal with unreliable measurements, we measure the stabilizer generators $O(d)$ times, where d is the code distance. In the 2D case, we imagine building up a 3D lattice from 2D time-slices, where each slice is a copy of the surface code lattice. We connect each slice to its preceding and succeeding slices by edges that link vertices with the same spatial position in the different time-slices. In this new lattice, the vertices store the change in the value of the corresponding stabilizer from the previous time-slice to the current time-slice. Errors on the physical qubits are edges in a particular time-slice and measurement errors are edges between different time-slices. Therefore, the syndrome in this case is a subset of the vertices of the 3D lattice. We can again map this problem to a minimum-weight matching problem, possibly using different edge weights for space-like and time-like edges to reflect different data qubit and measurement error probabilities.

Since the Blossom algorithm works for arbitrary graphs [31, 32], we can decode efficiently. To use this method on 3D surface codes, we build a 4D lattice from 3D time-slices in an analogous way.

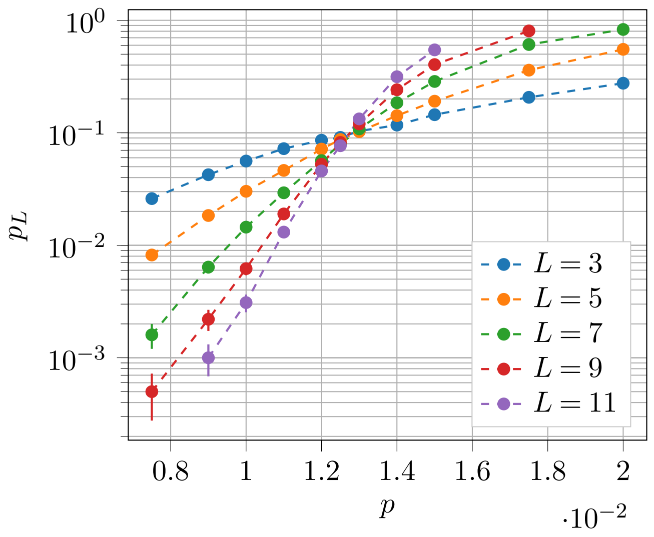
From the previous paragraph, we now understand that decoding Z-errors in 3D surface codes can be reduced to solving minimum-weight matching problems. We used an implementation of the Blossom algorithm in the library NetworkX [140] to estimate the Z-error threshold of 3D surface codes for perfect measurements and unreliable measurements. Specifically, we considered an error model where a Z-error is applied independently to each physical qubit with probability p, and each stabilizer measurement outcome is flipped with probability q. The surface code lattice we simulated was a $L \times L \times L$ cubic tessellation of the 3-torus. When $q = 0$ (perfect measurements), we observe an error threshold of approximately 2.9%, which is in agreement with the value found by Wang et al. using a mapping to a condensed matter model [141]. The data are shown in Figure 3.1a. When $q = p$ (unreliable measurements), we observe an error threshold of approximately 1.25%, as shown in Figure 3.1b. We expect a lower error threshold when stabilizer measurements are unreliable, but we note that the difference smaller than in the 2D case, where the error threshold for perfect measurements is approximately 10% [35] and the error threshold for unreliable measurements is approximately 1% [14].

As we stated previously, the results presented in Figure 3.1 are for surface codes defined on cubic tessellations of the 3-torus. However, we are also interested in the performance of MWPM for surface codes with boundaries, as in Chapter 4, we use such codes in our quantum computing architecture. We argue that the threshold value shown in Figure 3.1 are useful as an estimate of the error thresholds of codes with boundaries. We do not expect the boundaries to have a significant impact on the error threshold, because as the linear lattice size becomes large, the behaviour of the code will be dominated by the bulk. This intuition is correct for 2D surface codes, where the error threshold is similar whether or not the lattice has boundaries [139].

The surface code architectures we present in Chapter 4 use 3D surface codes defined on cubic lattices and tetrahedral-octahedral lattices (see Section 2.1.2). We expect the error thresholds for surface codes defined on different lattices to be different. In the 2D case, the (perfect measurement) error threshold of the triangular-lattice



a)



b)

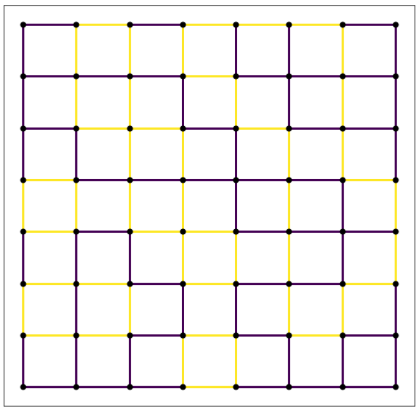
Figure 3.1: The error threshold of 3D surface code against phase-flip noise when measurements are perfect (3.1a) and when measurements are unreliable (3.1b). We plot the probability of a logical error, p_L, as a function of the physical error probability p (which is also the measurement error probability in 3.1b), for different values of L (the linear lattice size, or, equivalently, the code distance). Each data point represents 10,000 trials. In Figure 3.1a, we observe a crossing point (error threshold) at $p \approx 2.9\%$ and in Figure 3.1b, we observe an error threshold of $p \approx 1.25\%$. The error bars show the standard error of the mean: $\sqrt{p_L(1-p_L)/\eta}$, where η is the number of Monte Carlo trials. The data shown in Figure 3.1a were generated using ~ 900 CPU hours and the data shown in Figure 3.1b were generated using $\sim 11,000$ CPU hours. We acknowledge use of UCL supercomputing facilities (`https://www.ucl.ac.uk/research-it-services/services/research-computing-platforms`). The code that we used to produce these plots is available at the following URL: `https://github.com/MikeVasmer/Toric-Code-Matching`.

surface code is 6.6% [142], which is approximately two thirds of the square-lattice surface code error threshold. Similarly, we expect the error threshold of 3D surface codes defined on tetrahedral-octahedral lattices to be a fraction of the error threshold of the cubic-lattice surface code. This is because the X-stabilizers of the tetrahedral-octahedral surface code are higher weight than the X-stabilizers of the cubic surface code. We leave an investigation of the error threshold of tetrahedral-octahedral surface codes to future work. In Chapter 4, we use the error threshold values derived from Figure 3.1 as a rough estimate of the Z-error threshold of the 3D surface codes we use in our quantum computing architectures.

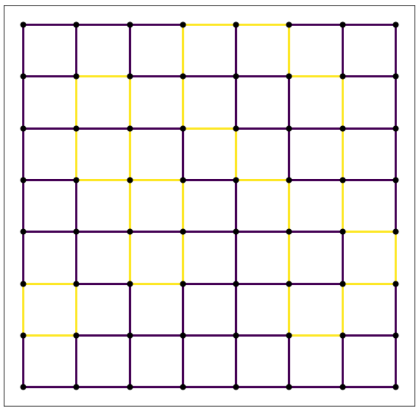
To conclude this section, we briefly discuss other algorithms that can be used to decode Z-errors in 3D surface codes. The Blossom algorithm is near-optimal and efficient, but its generalization to weighted graphs has (worst-case) runtime complexity $O\left(|V|^3|E|\right)$ [32], where V and E are the node-set and link-set of the graph. This runtime complexity is not ideal because we anticipate correcting errors after every operation in a fault-tolerant quantum computer. Therefore, a slow decoding algorithm would severely limit the processing speed of the computer. With this in mind, alternative decoding algorithms have been investigated for 2D surface codes. These include renormalization group (RG) decoders [143], cellular-automaton decoders [144, 145] and the Union-Find decoder [146]. In general, these approaches have better runtime complexity but worse performance than the MWPM algorithm. Given the similarity of the Z-error decoding problems in 2D and 3D, we anticipate that one could apply these decoders to 3D surface codes.

3.2 X-error decoding

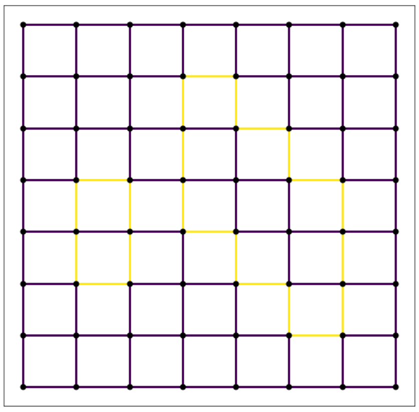
In this section, we propose a cellular automaton (CA) decoder for X-errors in surface codes defined on $(D \geq 3)$-dimensional lattices with boundaries. Our decoder builds on the Sweep Rule decoder, which was recently introduced by Kubica and Preskill [43]. The Sweep Rule decoder in turn is based on a (classical) cellular automaton called Toom's Rule [147]. This cellular automaton can be thought of as an extremely robust classical memory [147, 148, 149]. We give a brief overview of this aspect of Toom's rule here, because it shares many properties with our decoder. Our discussion closely follows [149]. Suppose we have a square lattice with spins placed on faces. Each spin can either be up or down (i.e. it is a classical bit). We associate



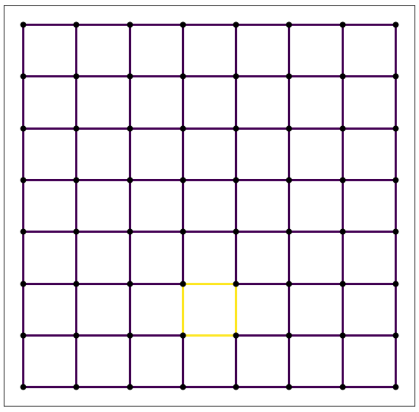
a)



b)



c)



d)

Figure 3.2: Applying Toom's rule to a randomly initalized lattice of (classical) spins. Spins are on faces, and an edge is bright (dark) if the spins on the faces that are part of the edge are the same (different). Figure 3.2a shows the initial state of the lattice, before we apply the rule. Figures 3.2b, 3.2c, and 3.2d show the state of the lattice after the rule has been applied one, three, and seven times, respectively.

check operators with edges of the lattice, where a check operator is satisfied if both the faces it is part of have the same value, and is unsatisfied otherwise. The dynamics of the rule are simple: at each time step, we flip the spin on a given face if its north and east check operators are unsatisfied. We note that we simultaneously update the values of all the spins before updating the values of the check operators.

Figure 3.2 shows Toom's rule being applied multiple times to a randomly initalized lattice of spins. We can also make the rule probabilistic, by stipulating that after we update the value of a spin, if the spin points up then we change it to down with probability p and if the spin points down we change it to up with probability q.

The remarkable behaviour of Toom's CA is made clear by making an analogy with the Ising model [147, 148, 149]. One can define a "magnetic field" $h = (p-q)/(p+q)$, a "temperature" $T = p+q$ and a "magnetization", M, which is the expectation value of any spin. Toom proved that two stable phases exist (one with $M > 0$ and one with $M < 0$) for non-zero values of T and/or h [147]. Figure 3.3 shows the phase diagram. Consequently, as long as the temperature and magnetic field are below critical values, we can use a system with dynamics described by Toom's rule to store a classical bit. In contrast, in a 2D Ising model, there is no region of the phase diagram with non-zero magnetic field where two stable phases exist. As observed by Bennett and Grinstein, the key reason for the difference in the behaviour of Toom's rule and the 2D Ising model is the irreversibility of Toom's rule [148].

Toom's rule is similar to a 3D surface code in the dual Kitaev picture. We recall that in this picture, we place qubits on faces, we associate Z-stabilizer generators with edges and we associate X-stabilizer generators with cells. The Z-stabilizer generators are analogous to the check operators in Toom's rule, and as decoding X-errors on their own is essentially a classical error-correction problem, we can imagine that a decoder based on Toom's rule may be very robust. Indeed, one can make an analogy between the temperature in Toom's rule and the physical qubit bit-flip error probability in a 3D surface code; and between the magnetic field in Toom's rule and the measurement error probability in a 3D surface code. The stability of Toom's rule implies that a 3D surface code decoder based on this rule should be resilient against physical qubit errors and measurement errors. This is exactly the behaviour we observe for our cellular automaton decoder in Section 3.2.3.

Topological code decoders based on cellular automata are attractive because such decoders process syndrome information locally. Therefore, decoders of this type are inherently parallelizable and could be implemented using specialist hardware near to the qubits themselves. In contrast, decoders such as MWPM require complex

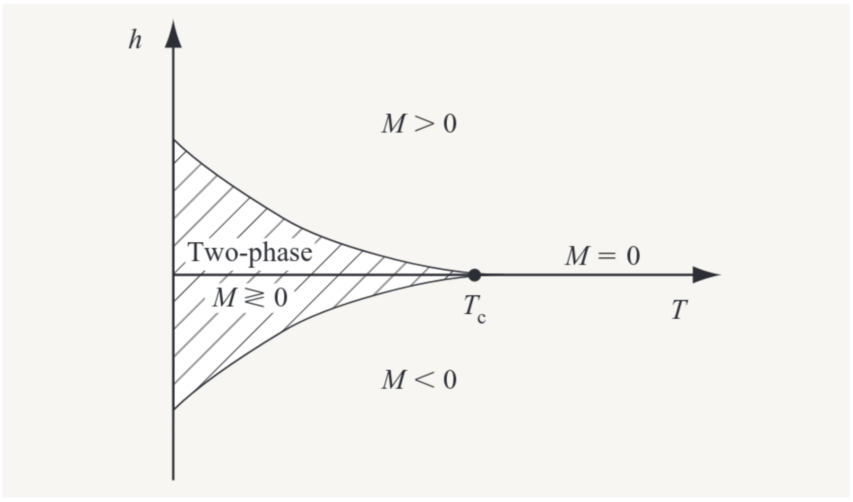


Figure 3.3: The phase diagram of a spin system with dynamics governed by Toom's rule. In contrast to equilibrium systems like the the Ising model, two stable phases exist for non-zero values of h (shaded region). Figure adapted from Grinstein [149].

processing of the entire syndrome. Also, since it is not currently known whether a passive quantum memory exists in fewer than four dimensions [47], a code requiring only local error correction may be the closest we can get to a passive quantum memory in 3D.

Previously, cellular automaton decoders have been proposed for 2D surface codes [150, 151, 144, 145], anyon systems [152], and 4D surface codes [28, 153, 151]. Recently, Kulkarni and Sarvepalli showed that one can use a decoder based on Toom's rule to correct errors in 3D surface codes defined on cubic lattice (with and without boundaries) [44]. However, we want a decoder that works on a wider variety of 3D lattices, because the transversal implementation of CCZ we presented in Chapter 2 requires 3D surface codes defined on non-cubic lattices.

The Sweep Rule decoder [43, 154] almost meets our requirements. This decoder is based on a CA that generalizes Toom's rule to a broad class of lattices. However, the Sweep Rule decoder does not work for lattices with boundaries, as we illustrate in Figure 3.4. In the remainder of this section, we adapt the Sweep Rule decoder to work for lattices with boundaries. First, we review the definition of the Sweep Rule cellular automaton. Secondly, we prove that a 3D surface code decoder based on this cellular automaton has a non-zero error threshold when measurements are perfect. Finally, we report numerical evidence of an error threshold when stabilizer measurements are unreliable, and we compare the performance of our decoder with

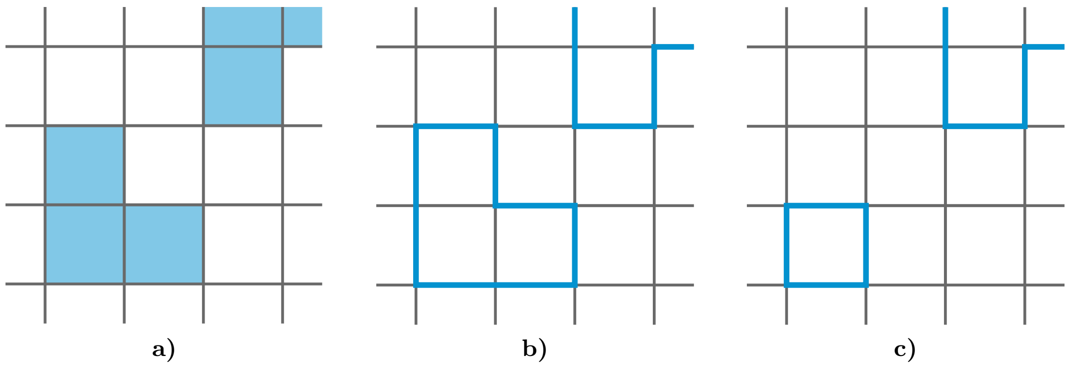


Figure 3.4: Errors that are not removed by the Sweep Rule decoder. We show a slice of a 3D surface code with qubits on faces and Z-stabilizer generators associated with edges. In this example, the Sweep Rule decoder is essentially the same as Toom's rule. For each face in the lattice, the Sweep Rule applies an X-operator to the qubit on a face if the north and east stabilizers of the face are unsatisfied. Figure 3.4a shows an X-error (blue faces) and Figure 3.4b shows the corresponding syndrome (blue edges). Figure 3.4c shows the same lattice after the application of the rule. We see that the Sweep Rule partially corrects the error in the bulk of the lattice, whereas the error on the boundaries does not change. This is because the corresponding syndrome does not contain both the north and the east edges of any face. To deal with this problem, we can run Toom's rule again, but using south and west edges instead. In the language of the Sweep Rule, this is equivalent to changing the sweep direction, as we explain in Section 3.2.1.

other approaches.

3.2.1 Sweep Rule with boundaries

In this section, we analyse the behaviour of the Sweep Rule for lattices with boundaries. We begin by defining basic concepts which we will use throughout the remainder of this chapter. Next, we state the conditions that a lattice must satisfy so that the Sweep Rule can be defined on it. Finally, we give a general definition of the Sweep Rule and illustrate how it works using a 2D example. Many of the definitions in this section are lightly modified versions of definitions that originally appeared in [43].

3.2.1.1 Preliminary definitions

To begin, we consider infinite regular lattices (we introduce boundaries later). Furthermore, we only consider simplicial lattices i.e. lattices formed by attaching D-dimensional simplices along their $(D-1)$-dimensional boundaries. We recall that a D-dimensional simplex is the D-dimensional analogue of a triangle. For example, a 0D simplex is a point, a 1D simplex is a line, a 3D simplex is a tetrahedron, and a 4D simplex is a tetrahedral prism. We confine our attention to simplicial lattices to simplify the error threshold proof in Section 3.2.2. However, the intuition we develop in the following sections can be extended to other lattices, as evidenced by the numerical results we present for non-simplicial lattices in Section 3.2.3.

Let $\mathcal{L}$ denote an infinite simplicial lattice. We use $\Delta_i(\mathcal{L})$ to denote the set of i-dimensional simplices (i-simplices) in $\mathcal{L}$. We define an $\mathbb{F}_2$-vector space, C_i, with a basis formed by the i-simplices of the lattice. There is a bijection between vectors in C_i and subsets of i-simplices in $\mathcal{L}$. This allows us to define (linear) boundary operators, $\partial_i : C_i \to C_{i-1}$, which we specify for all basis elements:

$$\partial_i \iota = \sum_{\nu \in \Delta_{i-1}(\iota)} \nu, \tag{3.2}$$

where $\Delta_{i-1}(\iota)$ is the set of all $(i-1)$-simplices in ι. We define the m-star of an i-cell ι, $\mathrm{St}_m(\iota)$, to be the set of all m-simplices in the neighbourhood of ι that contain ι (where $m \geq i$).

We also need a set of sweep directions, $\mathcal{D}$. Each sweep direction is a unit vector in $\mathbb{R}^D$ that is not orthogonal to any of the edges of $\mathcal{L}$. We group a lattice, $\mathcal{L}$, and a set of sweep directions, $\mathcal{D}$, in a pair $(\mathcal{L}, \mathcal{D})$. For a decoder based on the Sweep Rule to work, we need the pair $(\mathcal{L}, \mathcal{D})$ to satisfy certain conditions, which we state in Section 3.2.1.2. But before stating these conditions, we need to define some additional concepts.

We define a path $(u : v)$ between two vertices, u and v, to be a collection of edges $(u, w_1), \ldots, (w_n, v)$, where each w_i is a vertex in the lattice. The distance between two vertices, u and v, is naturally defined to be the length of the smallest path between them, that is $d(u,v) = \min_{(u:v)} |(u : v)|$. We extend this definition to finite subsets of vertices, U and V, in the obvious way: $d(U,V) = \min_{u \in U, v \in V} d(u,v)$. We define the diameter of a subset of vertices, $\mathrm{diam}(V)$, to be the maximum distance between any two vertices in V. And we define a D-dimensional ball of radius r centred at a vertex v to be

$$B_v(r) = \bigcup_{i=0}^{D} \{\iota \in \Delta_i(\mathcal{L}) \,|\, d(\iota, v) < r\} \tag{3.3}$$

We call a path causal (with respect to a sweep direction $\vec{\delta}_j \in \mathcal{D}$), if the sign of the inner product, $\vec{\delta}_j \cdot (w_l, w_{l+1})$, is the same for all edges in the path. We use the notation $(u \updownarrow v)_j$ to denote a causal path from u to v. We define the causal distance between two vertices to be the length of the shortest causal path between them (with respect to $\vec{\delta}_j$), that is:

$$d_j(u,v) = \min_{(u \updownarrow v)_j} |(u \updownarrow v)_j|. \tag{3.4}$$

It could be the case that no causal path exists between two vertices, in this case we set the causal distance between them to be infinite.

As noted in [43], a sweep direction induces a partial order over the vertices of the lattice. We say that a vertex, u, precedes another vertex, v, with respect to $\vec{\delta}_j$, $u \preceq_j v$, if a there exists a causal path from u to v and all edges in the path have positive inner product with $\vec{\delta}_j$, or $u = v$. This is equivalent to saying that v succeeds u, $v \succeq_j v$, which is defined analogously. We also extend this notation to simplices. We say that an i-simplex ι precedes a vertex v, $\iota \preceq_j v$, if all the vertices that are part of ι themselves precede v. We define the future (past) of a vertex v to be the set of all simplices that succeed (precede) v, that is

$$\begin{aligned} \uparrow_j (v) &= \bigcup_{i=0}^{D} \{\iota \in \Delta_i(\mathcal{L}) \mid \iota \succeq_j v\}, \\ \downarrow_j (v) &= \bigcup_{i=0}^{D} \{\iota \in \Delta_i(\mathcal{L}) \mid \iota \preceq_j v\}, \end{aligned} \tag{3.5}$$

where $\uparrow_j (v)$ and $\downarrow_j (v)$ denote the future and past of v, respectively.

Let $V \subseteq \Delta_0(\mathcal{L})$ denote a finite subset of vertices. We define the future of V to be $\uparrow_j (V) = \bigcap_{v \in V} \uparrow_j (v)$ and we define the past of V to be $\downarrow_j (V) = \bigcap_{v \in V} \downarrow_j (v)$ (both with respect to the sweep direction $\vec{\delta}_j$). Figure 3.5 shows an example. The (unique) j-supremum of V, $\sup_j V$, is the least element in $\uparrow_j (V)$. Similarly, the (unique) j-infimum of V, $\inf_j V$ is the greatest element in $\downarrow_j (V)$. We now come to an important concept, the causal diamond of a set of vertices, which is defined as follows:

$$\Diamond_j (V) = \uparrow_j (\inf_j V) \cap \downarrow_j (\sup_j V). \tag{3.6}$$

Having defined all the relevant concepts in terms of infinite regular lattices, we now introduce boundaries. We exclusively consider finite lattices that are restrictions

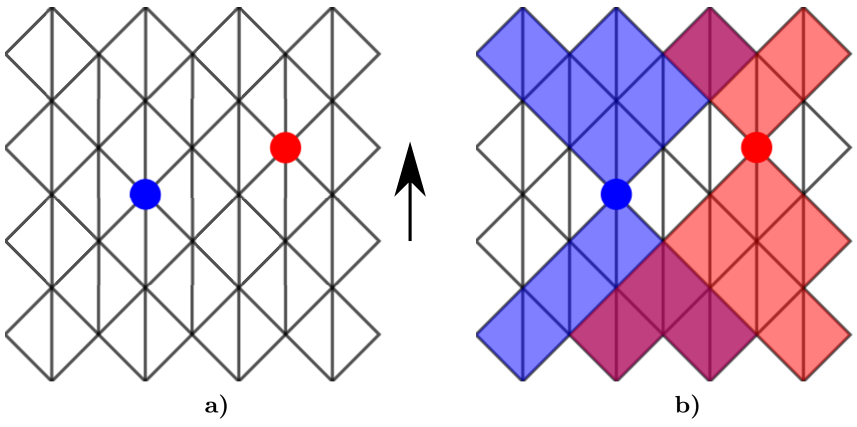


Figure 3.5: The future and past of a subset of vertices. Let a subset of vertices V contain the red and the blue vertices highlighted in Figure 3.5a. In Figure 3.5b, we show the future and the past of each of the vertices in V (red and blue shaded regions of the lattice). The sweep direction is illustrated by the arrow. The future (past) of V is the intersection of the futures (pasts) of its constituent vertices (purple shaded regions of the lattice).

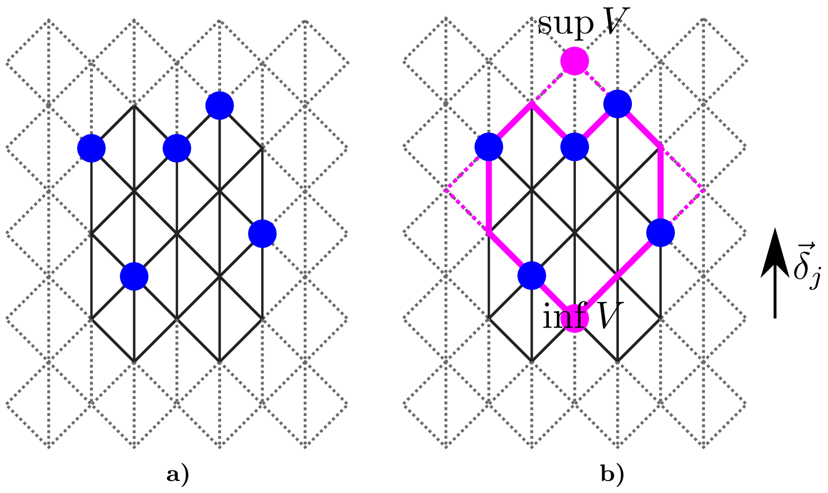


Figure 3.6: A causal region in a simplicial lattice with boundaries. In Figure 3.6a, we show a subset of vertices (blue circles), V, in a finite lattice (solid grey lines). The dashed grey lines represent the infinite lattice. In Figure 3.6b, we show the infimum and supremum of V with respect to the sweep direction $\vec{\delta}_j$. The infimum lies inside the finite lattice whereas the supremum is in the infinite lattice. The causal region of V consists of everything inside the solid pink lines (the dashed pink lines represent the parts of the causal diamond outside the finite lattice).

of infinite regular lattices i.e. we consider a connected subset of the simplices of the infinite lattice. This stipulation allows us to carry over all the definitions we made above without changes, except for the causal diamond. In a lattice with boundaries, for a given sweep direction $\vec{\delta}_j$, a subset of vertices, $V \subset \Delta_0(\mathcal{L})$, may not have a

j-infimum and/or a j-supremum, and in this case the causal diamond is ill-defined. Therefore, we define the causal region of a V with respect to $\vec{\delta}_j$, $\mathcal{R}_j(V)$, to be the causal diamond of V in the infinite lattice but restricted to the lattice with boundaries. Figure 3.6 shows a subset of vertices whose causal region in the finite difference is different from the corresponding causal diamond in the infinite lattice.

3.2.1.2 Lattice conditions

Consider a pair $(\mathcal{L}, \mathcal{D})$, where $\mathcal{L}$ is a D-dimensional simplicial lattice with boundaries and $\mathcal{D}$ is a set of sweep directions. We place ± 1 spins on the i-simplices of $\mathcal{L}$, where $2 \leq i \leq D$. We use ϵ to denote a subset of i-simplices whose spins are -1 and we refer to such a set as an error. We denote the i-boundary of an error ϵ by $\sigma = \partial_i \epsilon$. Given a vertex, $v \in \Delta_0(\mathcal{L})$, we denote the restriction of the i-boundary σ to the neighbourhood of v by $\sigma|_v$. That is, $\sigma|_v$ contains the $(i-1)$-simplices that are in σ and $\mathrm{St}_{i-1}(v)$. For a given vertex v, if $\sigma|_v$ is non-empty and $\sigma|_v \subset \uparrow_j (v)$, then we call the vertex a j-trailing vertex. In other words, a vertex is j-trailing if the restriction of the i-boundary to this vertex is non-empty and is also in the future of v with respect to the sweep direction $\vec{\delta}_j \in \mathcal{D}$. Finally, we assume that the lattice has some linear size L, and we define a local region of the lattice to be a connected subset of simplices whose diameter is at most a fraction of L.

We now state the conditions that the pair $(\mathcal{L}, \mathcal{D})$ must satisfy if one wants to define a Sweep Rule for the pair. We call these requirements causality conditions:

1. Every sweep direction $\vec{\delta}_j \in \mathcal{D}$ is not orthogonal to any of the edges of $\mathcal{L}$.

2. For any subset of vertices within a local region of $\mathcal{L}$, $V \subset \Delta_0(\mathcal{L})$, there exists a unique causal region of V, $\mathcal{R}(V)$, which the same for all sweep directions $\vec{\delta}_j \in \mathcal{D}$.

3. For any i-boundary $\sigma \in \operatorname{im} \partial_i$ contained within a local region of $\mathcal{L}$, the j-infimum of σ is contained in the (finite) lattice for at least one sweep direction, $\vec{\delta}_j$. In this case, there exists at least one j-trailing vertex, v, such that one can find a set of i-simplices $\varphi(v) \subseteq \mathrm{St}_i(v) \cap \uparrow_j (v)$ with the following properties:

   (a) $[\partial_i \varphi(v)]|_v = \sigma|_v$

   (b) $\mathcal{R}(\varphi(v)) = \mathcal{R}(\sigma|_v)$

The first two conditions are straightforward, but the third is a little harder to parse. Consider the case where spins are on faces and the corresponding boundary, σ, is a collection of edges. The trailing vertices in this case are a subset of the corners of σ. The two edges in σ that meet at one of these corners will both have a positive inner product with the sweep direction. Condition three guarantees that a j-infimum of σ is contained in the finite lattice, which means that at least one j-trailing vertex will exist. And the second part of condition three ensures that we can flip a subset of faces in the neighbourhood of such a trailing vertex so that the boundary of the -1 faces is moved in the sweep direction.

In order to prove that the Sweep Rule gives rise to a 3D surface code decoder with a threshold, we also require $(\mathcal{L}, \mathcal{D})$ to be locally Euclidean. This concept is captured in the following requirements:

1. For any ball $B_v(R)$ of radius R within a local region of $\mathcal{L}$ there exists a cover,

$$\bigcup_{u \in U} B_u(r) \supset B_v(R), \tag{3.7}$$

consisting of balls of radius $r < R$ indexed by $U \subset \Delta_0(\mathcal{L})$, such that

$$|U| < c_B (R/r)^D, \tag{3.8}$$

where D is the dimension of the lattice and c_B is a constant.

2. For any subset of vertices $V \in \Delta_0(\mathcal{L})$ within a local region of $\mathcal{L}$,

$$\operatorname{diam}(\mathcal{R}(V)) \le c_D \times \operatorname{diam}(V), \tag{3.9}$$

where $c_D \ge 1$ is a constant. In other words, the diameter of the causal region of V is upper bounded by c_D times the diameter of V.

3. For any pair of vertices, u and v, and sweep direction $\vec{\delta}_j \in \mathcal{D}$, if $u \preceq_j v$, then there exists a constant $c_P \ge 1$ such that

$$\max_{(u \updownarrow v)_j} |(u \updownarrow v)_j| \le c_P \times d(u, v), \tag{3.10}$$

that is, the length of the longest causal path between two vertices is upper

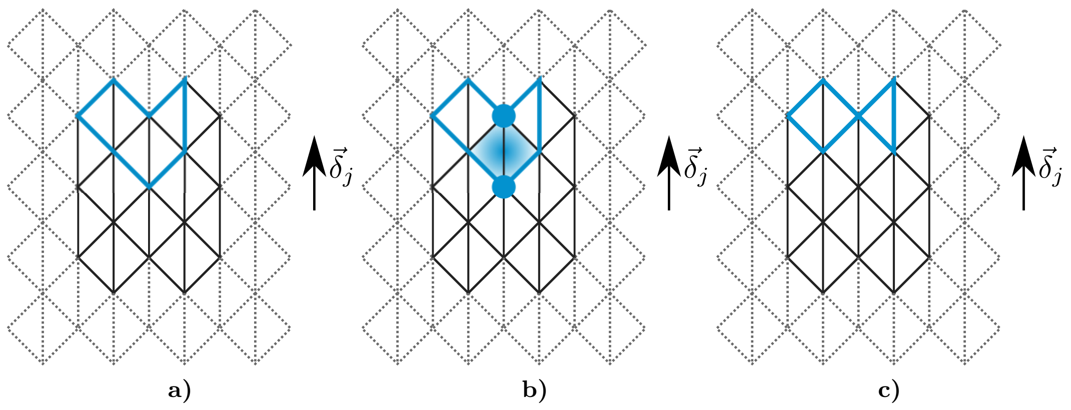


Figure 3.7: An example of how the Sweep Rule works. In Figure 3.7a, we show a 2-boundary σ (solid blue) in a 2D causal lattice with boundaries. We recall that this lattice (solid grey) is restriction of an infinite regular lattice, which is illustrated here using dashed grey lines. The sweep direction $\vec{\delta}_j$ points upwards. In Figure 3.7b, we highlight the j-trailing vertices with blue circles. When we apply the Sweep Rule to the top j-trailing vertex, nothing happens because there are no triangles in the future of the vertex whose boundary locally matches σ. But when we apply the Sweep Rule to the bottom j-trailing vertex, there are two triangles (shaded blue) in the future of the vertex whose (joint) boundary locally matches σ. Therefore, the Sweep Rule flips the spins on these triangles. Finally, in Figure 3.7c, we show the new 2-boundary after the application of the Sweep Rule.

bounded by c_P times the distance between them.

We emphasize that the three constants, c_B, c_D and c_P must all be independent of the size of the lattice.

3.2.1.3 Sweep Rule definition

Let $\mathcal{L}$ be a lattice with boundaries and let $\mathcal{D}$ be a set of sweep direction chosen such that the pair $(\mathcal{L},\mathcal{D})$ satisfy the causality conditions given in Section 3.2.1.2. Place spins on the i-simplices of $\mathcal{L}$ and let σ be the i-boundary of an error ϵ. For every vertex $v \in \Delta_0(\mathcal{L})$ and each sweep direction $\vec{\delta}_j \in \mathcal{D}$, the sweep rule is defined as follows:

Definition 2 (Sweep Rule [43])**.** If v is j-trailing, find a subset of neighbouring i-simplices $\varphi(v) \subset \uparrow_j (v)$ with an i-boundary that locally matches σ, i.e. $[\partial_i \varphi(v)]|_v = \sigma|_v$. Flip the spins on the i-simplices in $\varphi(v)$.

We emphasize that the Sweep Rule defined above is identical to the Sweep Rule introduced in [43]. Figure 3.7 shows an example of the Sweep Rule being applied to a 2-boundary. From this Figure, we see the similarity between the Sweep Rule and Toom's Rule. Indeed, the Sweep Rule on a square lattice with a sweep direction pointing south-west is the same as Toom's Rule. The utility of the Sweep Rule is that there are some lattices where a simple generalization of Toom's Rule does not remove certain i-boundaries, whereas the Sweep Rule removes them [43].

3.2.2 Proof of error threshold

In this section, we analyse the performance of a decoder based on the Sweep Rule when stabilizer measurements are perfect. We adapt the proof presented in [43] to show that our decoder has a non-zero error threshold for for errors with m-dimensional syndromes in surface codes with boundaries, where $m \geq 1$. We recall from Section 1.2.1, that a type-i surface code has qubits on i-cells, X-stabilizers that are associated with $(i-1)$-cells, and Z stabilizers that are associated with $(i+1)$-cells, where $i \in \{1 \ldots D-1\}$. In such a surface code, we say that X (Z) errors have m-dimensional syndromes, where m is the smallest dimensional lattice element (m-cell) to which Z (X) stabilizers are associated in either the primal or dual lattice. For example, consider type-1 3D surface codes. In the primal lattice, qubits are on edges (1-cells), X-stabilizers are associated with vertices (0-cells), and Z-stabilizers are associated with faces (2-cells) And in the dual lattice, qubits are on faces, X stabilizers are associated with cells (3-cells), and Z-stabilizers are associated with edges. Therefore, Z-errors in 3D surface codes have 0-dimensional syndromes and X-errors have 1-dimensional syndromes. Consequently, our decoder can be applied to X-errors in 3D surface codes, but not to Z-errors.

We begin by proving that the Sweep Rule has certain properties, before using these properties to show that a decoder based on the Sweep Rule has a non-zero error threshold. The structure of our proof closely follows the structure of the proof presented in [43]. Our main new contribution is to generalize the Sweep Rule properties Lemma to lattices with boundaries. Once this is accomplished, the remainder of the proof proceeds in much the same way as the original.

3.2.2.1 Sweep Rule properties

Before we consider the properties of the Sweep Rule, we prove the following lemma about causal regions.

Lemma 4 *For $U, V \subset \Delta_0(\mathcal{L})$, if $U \subseteq V$, then*

$$U \subseteq \mathcal{R}(U) \subseteq \mathcal{R}(V). \tag{3.11}$$

Proof. To show that $U \subseteq \mathcal{R}(U)$, we recall that $\mathcal{R}(U)$ is defined to be the restriction of $\uparrow_j (\inf_j U) \cap \downarrow_j (\sup_j U)$ to the finite lattice (where j is arbitrary by causality condition 2). We observe that both $\uparrow_j (\inf_j U)$ and $\downarrow_j (\sup_j U)$ contain U, so their intersection also contains U. If $U \subseteq V$, then either $\inf_j U = \inf_j V$ or $\inf_j U \succeq_j \inf_j V$. In either case, $\uparrow_j (\inf_j U) \subseteq \uparrow_j (\inf_j V)$. Likewise, if $U \subseteq V$, then either $\sup_j U = \sup_j V$ or $\sup_j U \preceq \sup_j V$. In either case $\downarrow_j (\sup_j U) \subseteq \downarrow_j (\sup_j V)$. Therefore, anything in the intersection of $\uparrow_j (\inf_j U)$ and $\downarrow_j (\sup_j U)$ will also be in the intersection $\uparrow_j (\inf_j V)$ and $\downarrow_j (\sup_j V)$, i.e. $\mathcal{R}(U) \subseteq \mathcal{R}(V)$. □

We now prove some facts about how i-boundaries behave when we repeatedly apply the Sweep Rule using the same sweep direction.

Lemma 5 *Let $\mathcal{L}$ be a lattice with boundaries and let $\mathcal{D}$ be a set of sweep direction chosen such that the pair $(\mathcal{L}, \mathcal{D})$ satisfy the causality conditions given in Section 3.2.1.2. Let ϵ be an error contained in a local region of $\mathcal{L}$, and let $\sigma = \partial_i \epsilon$ denote the i-boundary of the error. Choose a sweep direction $\vec{\delta}_j \in \mathcal{D}$. Suppose we apply the Sweep Rule simultaneously to every vertex of $\mathcal{L}$ between time steps T and $T+1$ for $T = 0, 1, \ldots$. Then the following properties hold:*

1. *(Support) The i-boundary at time T, $\sigma^{(T)}$, stays within the causal region of the original i-boundary, $\mathcal{R}(\sigma)$, that is*

$$\sigma^{(T)} \subseteq \mathcal{R}(\sigma) \quad \forall T. \tag{3.12}$$

2. *(Propagation) The causal distance between σ and any vertex v of $\sigma^{(T)}$ is at most T, that is*

$$d_j(v, \sigma) \leq T \quad \forall v \in \Delta_0\left(\sigma^{(T)}\right). \tag{3.13}$$

Proof. We first prove the support property by induction. This property is clearly true before the Sweep Rule has been applied (i.e. at time $T = 0$) because $\sigma^{(0)} = \sigma$.

Now we prove the inductive step from time T to time $T+1$. Between times T and $T+1$, depending on the sweep direction, the i-boundary may or may not contain j-trailing vertices. If the i-boundary does not contain any j-trailing vertices, then no spins are flipped and hence $\sigma^{(T+1)} = \sigma^{(T)} \subseteq \mathcal{R}(\sigma)$. Now, assume that the i-boundary contains at least one j-trailing vertex. Let $V^{(T)}$ denote the set of j-trailing vertices of the i-boundary at time T. For each j-trailing vertex $v \in V^{(T)}$, the Sweep Rule may find a subset of i-simplices, $\varphi^{(T)}(v)$, whose boundary locally matches the i-boundary, that is $[\partial_i \varphi^{(T)}(v)]|_v = \sigma^{(T)}|_v$. In this case, the spins on the i-simplices in $\varphi^{(T)}(v)$ are flipped so the i-boundary becomes

$$\sigma^{(T+1)} = \sigma^{(T)} + \sum_{v \in V^{(T)}} \partial_i \varphi^{(T)}(v). \tag{3.14}$$

By causality condition 3, $\mathcal{R}\left(\varphi^{(T)}(v)\right) = \mathcal{R}\left(\sigma^{(T)}|_v\right)$, which implies that $\mathcal{R}\left(\partial_i \varphi^{(T)}(v)\right) \subseteq \mathcal{R}\left(\varphi^{(T)}(v)\right) \subseteq \mathcal{R}\left(\sigma^{(T)}\right) \subseteq \mathcal{R}(\sigma)$. Combining this with Lemma 4 implies the following:

$$\begin{aligned} \sigma^{(T+1)} \subseteq \mathcal{R}\left(\sigma^{(T+1)}\right) &= \mathcal{R}\left(\sigma^{(T)} \cup \bigcup_{v \in V^{(T)}} \partial_i \varphi^{(T)}(v)\right), \\ &\subseteq \mathcal{R}\left(\mathcal{R}\left(\sigma^{(T)}\right) \cup \bigcup_{v \in V^{(T)}} \mathcal{R}\left(\varphi^{(T)}(v)\right)\right) \subseteq \mathcal{R}(\sigma). \end{aligned} \tag{3.15}$$

Next, we prove the propagation property, also by induction. For $T=0$, the propagation property holds because the Sweep Rule has not been applied so the causal distance between σ and any vertex of $\sigma^{(0)}$ is zero. Now we prove the inductive step from time $T-1$ to T. Either $\sigma^{(T)} = \sigma^{(T-1)}$ and the propagation property is trivially true or $\sigma^{(T)} \neq \sigma^{(T-1)}$. In this case, for every vertex $v \in \Delta_0\left(\sigma^{(T)}\right)$, either $v \in \Delta_0\left(\sigma^{(T-1)}\right)$ or there exists an edge between v and some vertex $u \in \Delta_0\left(\sigma^{(T-1)}\right)$. The second possibility occurs when, between times $T-1$ and T, the Sweep Rule flips a spin on an i-simplex containing an edge that links u and v. By invoking the triangle inequality, we see that

$$\begin{aligned} d_j(v,\sigma) &\leq d_j(v,u) + d_j(u,\sigma), \\ &\leq 1 + (T-1) = T. \end{aligned} \tag{3.16}$$

□

The final lemma we prove about the Sweep Rule concerns the behaviour of i-boundaries when we repeatedly apply the Sweep Rule and vary the sweep direction.

Lemma 6 *Let $\mathcal{L}$ be a lattice with boundaries and let $\mathcal{D}$ be a set of sweep direction chosen such that the pair $(\mathcal{L},\mathcal{D})$ satisfy the causality conditions given in Section 3.2.1.2. Let ϵ be an error contained in a local region of $\mathcal{L}$, and let σ denote the i-boundary of the error. Assume there are m sweep directions $\vec{\delta}_j \in \mathcal{D}$. For each $j = 0,1,\ldots,m-1$, suppose we apply the Sweep Rule (using sweep direction $\vec{\delta}_j$) simultaneously to all the vertices in the lattice for T_j^* time steps, where*

$$T_j^* \geq \max_{u\in\mathcal{R}(\sigma)} \max_{(u \updownarrow \sup_j \sigma)_j} |(u \updownarrow \sup_j \sigma)_j|. \tag{3.17}$$

That is, T_j^* is greater than or equal to the length of the longest causal path between any vertex in $\mathcal{R}(\sigma)$ and the j-supremum of $\mathcal{R}(\sigma)$. We use T to denote the time steps of the entire procedure. Therefore, T takes values in

$$\{0,1,\ldots,T_0^*,T_0^*+1,\ldots,T_0^*+T_1^*,\ldots,\sum_{j=0}^{m-1} T_j^*\}, \tag{3.18}$$

and the Sweep Rule is applied in between time steps, as in Lemma 5. Then the following properties hold:

1. *(Support) The i-boundary at time T, $\sigma^{(T)}$, stays within the causal region of the original i-boundary, $\mathcal{R}(\sigma)$, throughout the procedure.*
2. *(Propagation) The distance between any vertex of the i-boundary at time T and the original i-boundary is upper bounded by T.*
3. *(Removal) The i-boundary is removed by the end of the procedure (i.e. for times $T \geq \sum_j T_j^*$).*

Proof. We prove the support property by induction on the list of sweep directions. Consider the first sweep direction in the list $\vec{\delta}_0$. For all times $T \leq T_0^*$, we have $\sigma^{(T)} \subseteq \mathcal{R}(\sigma)$ by Lemma 5. Now, we prove the inductive step from j to $j+1$. Consider times $T' \in \{\sum_{l=0}^{j} T_l^*, (\sum_{l=0}^{j} T_l^*)+1,\ldots,\sum_{l=0}^{j+1} T_l^*\}$. Let $T_j = \sum_{l=0}^{j} T_l^*$. By Lemma 5, the i-boundary at all times T' is supported within the causal region of the i-boundary at time T_j, i.e. $\sigma^{(T')} \subseteq \mathcal{R}\left(\sigma^{(T_j)}\right)$. By assumption, $\sigma^{(T_j)} \subseteq \mathcal{R}(\sigma)$, and by Lemma 4

$$\mathcal{R}\left(\sigma^{(T_j)}\right) \subseteq \mathcal{R}(\mathcal{R}(\sigma)) = \mathcal{R}(\sigma), \tag{3.19}$$

which implies that $\sigma^{(T')} \subseteq \mathcal{R}(\sigma)$ for all T'.

Next we prove the propagation property, also by induction. Consider the causal distance between any vertex $v \in \Delta_0\left(\sigma^{(T)}\right)$ and the original i-boundary, where $T \in \{0,\ldots,T_0^*\}$. By Lemma 5 this causal distance is upper bounded by the number of steps that the Sweep Rule has been applied, i.e. for any v we have

$$d_0(v,\sigma) \leq T. \tag{3.20}$$

As the causal distance upper bounds the regular distance, the propagation property is true for $T \leq T_0^*$. Next, we prove the inductive step from j to $j+1$. Consider times $T' \in \{\sum_{l=0}^{j} T_l^*, (\sum_{l=0}^{j} T_l^*)+1,\ldots,\sum_{l=0}^{j+1} T_l^*\}$. Let $T_j = \sum_{l=0}^{j} T_l^*$. We assume that the distance between any vertex of the i-boundary at time T_j and the original i-boundary is upper bounded by T_j. For any vertex $u \in \Delta_0\left(\sigma^{(T')}\right)$, we have

$$d(u,\sigma) \leq d(u,\sigma^{(T_j)}) + d(w,\sigma), \tag{3.21}$$

where $w \in \Delta_0\left(\sigma^{(T_j)}\right)$ is the vertex that minimizes $d(u,\sigma^{(T_j)})$. The distance between u and $\sigma^{(T_j)}$ is upper bounded by the causal distance between them, so

$$\begin{aligned} d(u,\sigma) &\leq d_{j+1}(u,\sigma^{(T_j)}) + d(w,\sigma), \\ &\leq (T' - T_j) + T_j = T', \end{aligned} \tag{3.22}$$

where the inequality between the first and second lines holds by Lemma 5 and the inductive assumption.

To prove the removal property, we define the m-tuple of functions $(f_0^\sigma(T), f_1^\sigma(T),\ldots,f_{m-1}^\sigma(T))$, where

$$f_j^\sigma(T) = \begin{cases} \max_{u\in\mathcal{R}\left(\sigma^{(T)}\right)} \max_{(u \updownarrow \sup_j \sigma)_j} |(u \updownarrow \sup_j \sigma)_j| & \inf_j \sigma \in \Delta_0(\mathcal{L}), \\ \infty & \inf_j \sigma \notin \Delta_0(\mathcal{L}), \end{cases} \tag{3.23}$$

for $j \in \{0, 1, \dots, m-1\}$. If $\inf_j \sigma$ is contained within the finite lattice, $f_j^\sigma(T)$ is the length of the longest causal path between the supremum of the original i-boundary and any vertex within the causal region of the i-boundary at time T. As we noted in Section 3.2.1.1, for certain i-boundaries and sweep directions, $\sup_j \sigma$ may not be contained in the finite lattice. However, because we consider lattices that are finite regions of regular infinite lattices, $\sup_j \sigma$ will always exist in the infinite lattice for any σ.

We now show that at least one $f_j^\sigma(T)$ is a non-increasing function of T. By causality condition 3, $\inf_j \sigma$ is contained in the finite lattice for at least one of the sweep directions, say $\vec{\delta}_j$. In this case, $f_j^\sigma(T)$ is finite, and is upper bounded by $f_j^\sigma(0)$, the maximum causal distance between any vertex of the causal region of the original i-boundary and the supremum of the original i-boundary. This is because the causal region of the i-boundary at time T is contained in the causal region of the original i-boundary for all T (Sweep Rule support property).

Next, we define a new function

$$g_j^\sigma(T) = \max_{u \in \Delta_0\left(\sigma^{(T)}\right)} \max_{(u \updownarrow \sup_j \sigma)_j} |(u \underset{j}{\updownarrow} \sup \sigma)_j|, \tag{3.24}$$

which is equal to the longest causal path between any vertex of the i-boundary at time T and the supremum of the original i-boundary. The presence of $\inf_j \sigma$ in the finite lattice implies that any vertex $v \in \Delta_0(\sigma)$ that maximizes $g_j^\sigma(T)$ will have the following properties. First, v will be j-trailing, and secondly, there will exist a subset of neighbouring i-simplices $\varphi(v) \subset \uparrow_j (v)$ with a boundary that locally matches $\sigma^{(T)}$. Suppose we apply the Sweep Rule using sweep direction $\vec{\delta}_j$ between times T and $T+1$. Then the i-boundary will be modified in the neighbourhood of v. Specifically, v will not be part of $\sigma^{(T+1)}$, but other vertices in the neighbourhood of v will be part of $\sigma^{(T+1)}$, and these vertices will all be a shorter causal distance from $\sup_j \sigma$ than v was. Therefore,

$$g_j^\sigma(T+1) < g_j^\sigma(T). \tag{3.25}$$

Depending ordering of the sweep directions, it is possible that $g_j^\sigma(T) > g_j^\sigma(0)$, but we also have that $g_j^\sigma(T) \leq f_j^\sigma(T) \leq f_j^\sigma(0)$ for all T. This fact combined with Equation 3.25 implies that if we apply the Sweep Rule (with sweep direction $\vec{\delta}_j$) for time $T \geq f_j^\sigma(0) = T_j^*$, then the i-boundary will be removed. During the procedure described in Lemma 6, we apply the Sweep Rule T_j^* times for each sweep direction, respectively. Therefore, the i-boundary is sure to be removed at some point during the procedure. □

3.2.2.2 SweepRule decoder

In this Section, we detail a surface code decoder that is based on the Sweep Rule. We consider D-dimensional surface codes defined on causal lattices with boundaries, where $D \geq 3$. For concreteness, we consider tessellations the D-dimensional hypercube with linear lattice size L. We pick a number $i \in \{2, \dots, D-1\}$ and we place a qubit on each i-simplex in the lattice. We associate Z-stabilizers with $(i-1)$-simplices and we associate X-stabilizers with $(i+1)$ simplices. We consider a noise model where the bit-flip channel (Equation 1.3) is applied independently to each qubit, i.e. with probability p each qubit experiences a bit-flip (X) error. Consider an error, ϵ, generated according to this noise model. The error syndrome of ϵ is $\partial_i \epsilon = \sigma$, an i-boundary in the lattice. Decoding the error syndrome is the problem of estimating ϵ (up to a stabilizer) given σ. We construct an algorithm based on the Sweep Rule to solve this problem, which is described using pseudocode in Algorithm 1. We note that for $D \geq 4$, the Z-error decoding problem is analogous to the X-error decoding problem, so the SweepRule decoder can be used to correct both X and Z-errors in $(D \geq 4)$-dimensional surface codes.

Algorithm 1 SWEEPRULE DECODER

```
Input: $X$-error syndrome $\sigma \in \operatorname{im}\partial_i$, sweep directions $\mathcal{D}$, times $\mathcal{T}$
Output: Correction $\zeta \subseteq \Delta_i(\mathcal{L})$
  $T \leftarrow 0$
  $\sigma^{(0)} \leftarrow \sigma$
  $\zeta \leftarrow \emptyset$
  $T_{max} \leftarrow \textsc{Sum}(\mathcal{T})$
  while $\sigma^{(T)} \neq 0$ and $T < T_{max}$ do
      for $j \leftarrow 0$ to $|\mathcal{D}| - 1$ do
          $T' \leftarrow 0$
          $\vec{\delta} \leftarrow \mathcal{D}[j]$
          $T^* \leftarrow \mathcal{T}[j]$
          while $T' \leq T^*$ do
              Apply the Sweep Rule with sweep direction $\vec{\delta}$ simultaneously to every
              vertex of $\mathcal{L}$ to get $\zeta^{(T)}$, the $i$-simplices flipped by the rule.
              $\sigma^{(T+1)} \leftarrow \sigma^{(T)} + \partial_i \zeta^{(T)}$
              $\zeta \leftarrow \zeta \ominus \zeta^{(T)}$          ▷ $\ominus$ denotes the symmetric difference of sets
              $T' \leftarrow T' + 1$
              $T \leftarrow T + 1$
          end while
      end for
  end while
  if $T = T_{max}$ then
      return FAIL
  else
      return $\zeta$
  end if
```

There are two ways the SweepRule decoder can fail. Either the i-boundary is not removed and the algorithm returns FAIL or the algorithm returns a correction, ζ, such that the product of the original error, ϵ, and ζ is a logical X-operator. We now prove that the probability that the SweepRule decoder fails is exponentially suppressed in the linear size of the lattice, as long as the physical qubit error probability is below some non-zero threshold error probability. This statement is formalized in the following theorem.

Theorem 4 *Consider a family of lattices that tessellate the D-dimensional hypercube $\{\mathcal{L}\}$, with growing linear size L. Choose a set of sweep directions $\mathcal{D}$ such that each pair $(\mathcal{L}, \mathcal{D})$ satisfies the causality conditions and the local Euclidean conditions detailed in Section 3.2.1.2. Pick an integer $i \in \{2, \ldots, D-1\}$. Define a D-dimensional surface code on each $\mathcal{L}$, where qubits are placed on i-simplices and Z-stabilizers are associated with $(i-1)$-simplices. Assume an error model where at each time step, a* *bit-flip error is applied independently to each qubit with probability p. Then, there exists a constant $p_{th} > 0$, such that if $p < p_{th}$ the probability that the SweepRule decoder fails to correct the error is $O\left(\left(\frac{p}{p_{th}}\right)^{L}\right)$.*

To prove Theorem 4 we adapt the proof presented in [43], which itself builds on work in [33, 150, 155]. The proof strategy is as follows. First, we show that errors can be decomposed into chunks that are reasonably small and well-separated from each other. Next, we show that the SweepRule decoder removes these chunks of the error independently from each other in a bounded amount of time. Then, we show that the probability of an error containing large chunks is suppressed in the size of the chunks. As the maximum size chunk that the SweepRule decoder can correct grows with the lattice size L, we can suppress the probability of a logical error by increasing L.

3.2.2.3 Chunk decomposition

Let $\epsilon \subset \Delta_i(\mathcal{L})$ be an error in a D-dimensional surface code. A level-0 chunk is a single error and a level-t chunk is the disjoint union of two level-$(t-1)$ chunks that are contained with a sufficiently small region of $\mathcal{L}$. More specifically, a level-0 chunk $E^{[0]}$ is a single element of ϵ and a level-t chunk is recursively defined to be $E^{[t]} = E_1^{[t-1]} \sqcup E_2^{[t-1]}$ such that $\mathrm{diam}(E^{[t]}) \leq Q^t/2$ for some constant Q. We use $\sqcup$ to denote the disjoint union of two sets, i.e. the union of two sets with an empty intersection. The level-t error $E_t \subseteq \epsilon$ is defined to be the union of all level-t chunks, i.e. $E_t = \bigcup_i E_i^{[t]}$. We have the following inclusions:

$$\epsilon = E_0 \supseteq E_1 \supseteq \ldots \supseteq E_m \supsetneq E_{m+1} = \emptyset, \tag{3.26}$$

where m is always finite for any finite ϵ. Given the above, an error ϵ can be decomposed as follows:

$$\epsilon = F_0 \sqcup F_1 \sqcup \ldots \sqcup F_m, \tag{3.27}$$

where $F_t = E_t \setminus E_{t+1}$. A subset $M \subseteq \epsilon$ is said to be an l-connected component if, for any $M_1, M_2 \neq \emptyset$ such that $M = M_1 \sqcup M_2$ we have $d(M_1, M_2) \leq l$. In what follows, we need the following lemma concerning connected components:

Lemma 7 (Connected Components [33]) *Let $\epsilon \subset \Delta_i(\mathcal{L})$ be an error with disjoint decomposition $\epsilon = F_0 \sqcup F_1 \ldots$ and let $Q \geq 6$ be a constant. Let $M \subseteq \epsilon$ be a Q^t-*

connected component of F_t. Then, diam$(M) \leq Q^t$ and $d(M, E_i \setminus M) > Q^{t+1}/3$.

Lemma 7 gives us both an upper bound on the size of any Q^t-connected component of the error and a lower bound on the separation of the Q^t-connected component from the rest of the error. For a proof, see [33, 43]. Next, we use Lemma 7 to prove that the SweepRule decoder removes any level-t chunk of an error as long as t is smaller than a certain value.

Lemma 8 *Let $\epsilon \subset \Delta_i(\mathcal{L})$ be an error with disjoint decomposition $\epsilon = F_0 \sqcup F_1 \ldots$. Choose constants $Q = 6|\mathcal{D}|c_P c_D$ and $m^* = \lceil \log_Q(L/c_D) \rceil$, where L is the linear lattice size, $\mathcal{D}$ is the set of sweep directions, and c_D and c_P are lattice-dependent constants (see locally Euclidean conditions 2 and 3). Suppose we apply the Sweep Rule for $T_t = c_P c_D Q^t$ time steps in each of the sweep directions $\vec{\delta}_j \in \mathcal{D}$. Then for any Q^t-connected component M of F_t where $t < m^*$, the corresponding part of the i-boundary $\partial_i M$ will be removed by the end of the procedure. Furthermore, $\partial_i M$ is removed independently from the rest of the i-boundary.*

Proof. Consider some Q^t-connected component M of F_t. Given that $t < m^* = \lceil \log_Q(L/c_D) \rceil$, we have

$$\text{diam}(\mathcal{R}(M)) \leq c_D \times \text{diam}(M) \leq c_D Q^t < c_D Q^{m^*} \leq L, \tag{3.28}$$

where the first inequality holds because the lattice is locally Euclidean (condition 3). Equation 3.28 tells us that M is contained within a local region of $\mathcal{L}$, which implies that we can apply Lemma 6 to $\sigma = \partial_i M$.

First, we show that σ is removed by the procedure described in Lemma 8. Lemma 6 states that σ will be removed by the Sweep Rule if we sweep in each direction $\vec{\delta}_j \in \mathcal{D}$ for time T_j^*, where

$$T_j^* = \max_{u \in \mathcal{R}(\sigma)} \max_{(u \updownarrow \sup_j \sigma)_j} |(u \updownarrow \sup_j \sigma)_j|, \tag{3.29}$$

the longest causal path between any vertex of σ and the j-supremum of σ. For any M, we have $\sigma = \partial_i M \subset M$, so by Lemma 4 $\mathcal{R}(\sigma) \subseteq \mathcal{R}(M)$. Therefore, to get a bound on T_j^*, we bound

$$\max_{u\in\mathcal{R}(M)} \max_{(u\updownarrow \sup_j M)_j} |(u \updownarrow \sup_j M)_j|. \tag{3.30}$$

Due to the lattice being locally Euclidean (conditions 2 and 3) we have

$$\begin{aligned} \max_{u\in\mathcal{R}(M)} \max_{(u\updownarrow \sup_j M)_j} &|(u \updownarrow \sup_j M)_j|, \\ &\leq \max_{u\in\mathcal{R}(M)} c_P \times d(u, \sup_j M), \\ &\leq c_P \times \operatorname{diam}(\mathcal{R}(M)), \\ &\leq c_P c_D \times \operatorname{diam}(M). \end{aligned} \tag{3.31}$$

Lemma 7 tells us that $\operatorname{diam}(M) \leq Q^t$, which combined with Equation 3.31 gives us the following bound on T_j^*:

$$T_j^* \leq c_P c_D Q^t. \tag{3.32}$$

Next, we show that $\sigma = \partial_i M$ is removed independently of the rest of the i-boundary $\partial_i \epsilon \setminus \sigma$. Lemma 6 tells us that the distance of propagation for both σ and $\partial_i\epsilon \setminus \sigma$ is upper bounded by the total runtime of the procedure $T = \sum_j T_j^* = |\mathcal{D}| c_P c_D Q^t$. By Lemma 7, $d(\sigma, \partial_i \epsilon \setminus \sigma) \geq Q^{t+1}/3$, so in order for σ and $\partial_i\epsilon \setminus \sigma$ to remain separated during the procedure, we need $T = |\mathcal{D}| c_P c_D Q^t \leq Q^{t+1}/6$. This is satisfied for any $Q \geq 6|\mathcal{D}| c_P c_D$. □

To prove Theorem 4, we need one more ingredient. Using the van den Berg and Kesten inequality [156], it was shown in Ref. [43] that the probability of an level-t chunk occurring in a randomly chosen error $\epsilon \subseteq \Delta_i(\mathcal{L})$ is suppressed doubly exponentially in t. That is

$$\text{pr}(\text{level-}t \text{ chunk in } \epsilon) \leq |\Delta_0(\mathcal{L})| \lambda^{-2} \left(\frac{p}{p_{th}}\right)^{2^t}, \tag{3.33}$$

where $\lambda = (2Q)^D c_B$, D is the dimension of $\mathcal{L}$ and the threshold error probability is

$$p_{th} = \left(\lambda^2 \max_{v\in\Delta_0(\mathcal{L})} |\text{St}_i(v)|\right)^{-1}. \tag{3.34}$$

Proof. (Proof of Theorem 4). Lemma 8 tells us that the SweepRule decoder removes any Q^t-connected component M of F_t by time $T = |\mathcal{D}|c_P c_D Q^t$ as long as $t < m^* = \lceil \log_Q(L/c_D) \rceil$. In addition, each M is removed independently of the rest of the error when $Q = 6c_P c_D|\mathcal{D}|$, which is independent of the linear lattice size L. Finally, when $t < m^*$ we have $\mathrm{diam}(\mathcal{R}(M)) < L$ (Equation 3.28), and as M stays within $\mathcal{R}(M)$ throughout the procedure (Lemma 6), no correction can implement a non-trivial logical operator. Suppose we run the SweepRule decoder for time $T = |\mathcal{D}|c_P c_D Q^{(m^*-1)} + 1 = O(L)$. Then the SweepRule decode will successfully correct any error that does not contain level-m^* or higher chunks. When $p < p_{th}$, the probability of a of a level-t chunk being part of the error is doubly exponentially suppressed in t (Equation 3.34). As $m^* = O(\log L)$, the probability that the SweepRule decoder fails to correct the error is $O\left(\left(\frac{p}{p_{th}}\right)^L\right)$ when $p < p_{th}$. □

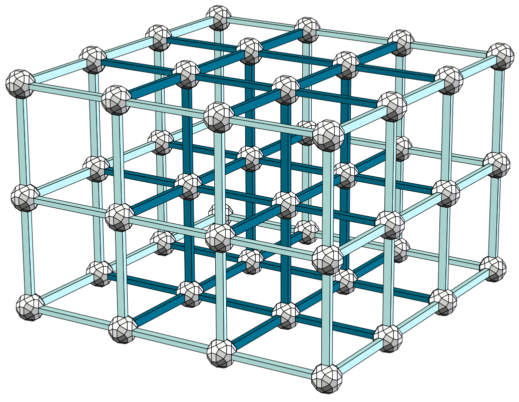
This concludes our proof that the SweepRule decoder has a non-zero error threshold. Let us now consider some example 3D surface codes, and see what error threshold our proof predicts for them. First, consider 3D surface codes defined on cubic lattices with boundaries (with qubits on faces and Z-stabilizer generators on edges). We can use the SweepRule decoder to correct X-errors in this code. We note that the cubic lattices are locally Euclidean, with parameters $c_D = c_P = 2$ and $c_B = 1$. In addition, $\max_{v \in \Delta_0(\mathcal{L})} |\mathrm{St}_2(v)| = 8$. Plugging these values into Equation 3.34 gives a threshold value $\approx 10^{-17}$. In the next Section, we will see that the real value is $\approx 10^{-1}$, which illustrates the importance of using simulations to get accurate estimates of the error threshold. The second code family we consider is surface codes defined on rhombic-dodecahedral lattices (see Figure 2.6). These lattices are locally Euclidean, with parameters $c_D \leq 3$, $c_B = 1$, $c_P = 1$ and $\max_{v \in \Delta_0(\mathcal{L})} |\mathrm{St}_2(v)| = 12$. Plugging these values into Equation 3.34 gives a threshold value $\approx 10^{-16}$, whereas the real value is $\approx 10^{-1}$ (as we show in the next section).

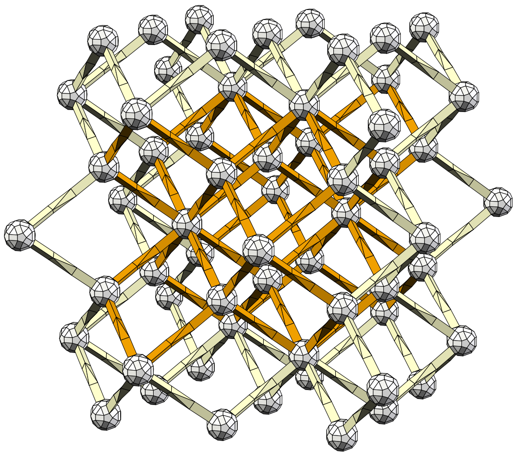
We note that neither the cubic lattice nor the rhombic-dodecahedral lattice is a simplicial lattice. One may worry, therefore, that the proof in the previous section will not apply to these lattices because we assumed a simplicial lattice structure. However, one can embed a rhombic-dodecahedral lattice in a simplicial lattice, the body-centred cubic (bcc) lattice. We recall that the bcc lattice is a lattice of cubes with additional vertices at the centres of the cubes, where each additional vertex is connected to the vertices of the cube it is at the centre of. Let us use a Cartesian coordinate system where the vertices of the cubes are located at integer coordinates and the vertices at the centres of cubes are located at half-integer coordinates. Then, one can construct a rhombic-dodecahedral lattice from a bcc lattice by deleting the vertices with odd half-integer coordinates (i.e. (i,j,k) such that $2(i+j+k)$ is odd) and deleting the edges between integer coordinates. Clearly, we can also embed a cubic lattice in a bcc lattice (just delete all half integer coordinates). The closeness of the cubic lattice and the rhombic-dodecahedral lattice to simplicial lattices gives us confidence that it should be possible to prove a non-zero error threshold for these lattices as well.

We emphasize that our proof only applies when stabilizer measurements are perfect. As we have said before in this thesis, in an actual quantum computer, measurement results will be unreliable. In general, the existence of a threshold when measurements are perfect does not imply that a threshold exists for the same decoder when measurements are unreliable. However, the stability of Toom's rule against a form of measurement noise (see Figure 3.3) gives us hope that the SweepRule decoder should work when for unreliable measurements. Indeed, this is what we observe in the simulations described in the next section. In future work, we plan to prove that the SweepRule decoder has a threshold when stabilizer measurements are unreliable. A similar result has recently been proved for a family of quantum codes based on expander graphs [157].

3.2.3 Numerical error threshold estimates

We implemented the SweepRule decoder in C++ and applied the decoder to 3D surface codes defined on lattices from the family of stacked 3D surface codes we described in Section 2.1.3. We recall that the surface codes in our family are defined on cubic lattices and rhombic-dodecahedral lattices. We use the picture where qubits are on faces, Z-stabilizer generators are associated with edges and X-stabilizer generators are associated with cells. The codes in our family are parameterized by their code distance, d, which is equal to $L-1$ where L is the linear dimension of the corresponding (Kitaev picture) lattice. Figure 3.8a shows the $d=3$ cubic lattice and Figure 3.8b shows the $d=3$ rhombic-dodecahedral lattice. We note that the cubic surface code is an example of $\mathcal{SC}_g$ in Table 2.1 and the rhombic-dodecahedral surface code is an example of $\mathcal{SC}_r$ or $\mathcal{SC}_b$ in Table 2.1 (these codes are the same up





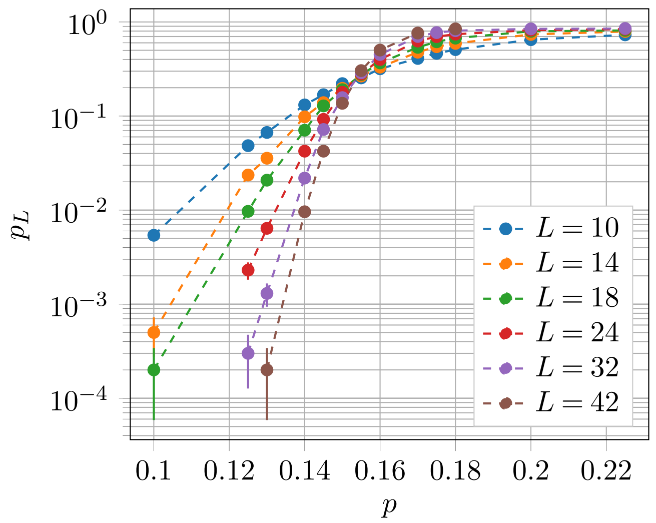
a) **b)**

Figure 3.8: Lattices of the codes described in Section 2.1.3 shown in the Kitaev picture (qubits on faces, Z-stabilizer generators on edges, X-stabilizer generators on cells). Figure 3.8a shows the $d = 3$ cubic lattice and Figure 3.8b shows the $d = 3$ rhombic-dodecahedral lattice, where $d = L - 1$ (d is the code distance and L is the linear size of the lattice). The faded edges are shown to illustrate how these lattices can be embedded in infinite lattices.

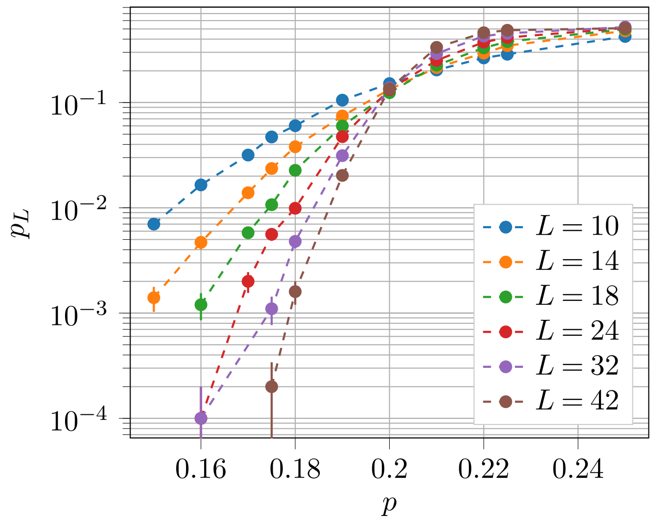
to a rotation of the lattice).

To estimate the error threshold of the two code families, we carried out Monte Carlo simulations to estimate the probability of a logical error, p_L, as a function of the physical error rate, p, for codes with growing linear size L. We used an error model where a bit-flip is applied independently to each physical qubit with probability p. In each case, we used eight sweep directions $\{(i, j, k) | i, j, k \in \{-1, 1\}\}$. One can verify that the lattices we consider combined with this set of sweep directions satisfy the causality conditions we detailed in Section 3.2.1.2 (the lattices are trivially locally Euclidean). We observe an error threshold of $\approx 15.5\%$ for cubic surface codes and we observe an error threshold of $\approx 20\%$ for rhombic-dodecahedral surface codes. The data are shown in Figure 3.9. We note that in each case, after applying the error channel we applied the sweep rule to every vertex $32 \times L$ times. This is in agreement with the scaling of the decoder runtime that we derived during the proof of Theorem 4. We changed the sweep direction after every L sweeps, meaning that each direction was swept four times.

The main difference between the two code families is that in cubic surface codes certain errors exist that are never removed by the SweepRule decoder (see Figure 3.10 for an example), whereas in rhombic-dodecahedral surface codes no



a)



b)

Figure 3.9: Error threshold plots for cubic surface codes (3.9a) and rhombic-dodecahedral surface codes (3.9b). In both cases, we apply the bit-flip channel independently to each of the physical qubits in the code and then we use the SweepRule decoder (Algorithm 1) to attempt to correct the error. In each subfigure, we plot the logical error probability, p_L, as a function of the physical error probability, p, for codes of increasing linear size, L. Each data point is the average of 10,000 trials. The error threshold is the value of p where these curves intersect. We observe an error threshold of $\approx 15.5\%$ for cubic surface codes and an error threshold of $\approx 20.0\%$ for rhombic-dodecahedral surface codes. The error bars show the standard error of the mean: $\sqrt{p_L(1-p_L)/\eta}$, where η is the number of Monte Carlo trials. The data in this Figure were generated using $\sim 1,800$ CPU hours. We acknowledge use of UCL supercomputing facilities (`https://www.ucl.ac.uk/research-it-services/services/research-computing-platforms`).

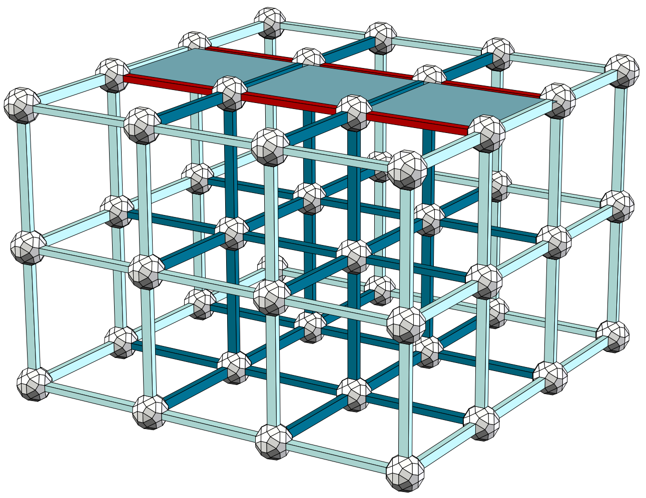


Figure 3.10: An example of a persistent syndrome in a 3D surface code. The filled-in faces have experienced bit-flip errors. The corresponding syndrome is shown in red. The SweepRule has no effect for this syndrome, intuitively because it has no corners. Persistent syndromes of this type have previously been observed by Breuckmann et al. [151].

such errors exist. However, these problematic syndromes are caused by specific errors (high-level chunks) with support $\geq L$, so we don't expect them to prevent the cubic surface codes from having an error threshold. We could design a subroutine to remove these errors once the main decoder has finished, as this problem maps straightforwardly onto a minimum-weight matching problem. We expect that this subroutine may improve the logical error probability but we would not expect it alter the error threshold. Therefore, we leave this optimization to future work.

We expect our decoder to be robust against measurement errors because of the similarity of the Sweep Rule CA to Toom's rule. This robustness is similar in spirit to the single-shot error correction property of codes such as the gauge colour code (see Section 1.3.3). In regular single-shot error correction, we use metachecks (consistency checks on the values of the stabilizers) to diagnose errors in the syndrome [51]. Hence, when a code has single-shot error correction, it is not necessary to repeat stabilizer measurements $O(d)$ times to deal with measurement errors. In the case of our SweepRule decoder, we make no use of metachecks. Instead, we simply apply the decoder without worrying about incorrect measurement results and we still observe a threshold. However, there seems to be a relationship between this kind of resilience to measurement errors and the single-shot error correction studied in [51]. In a 3D surface code, a loop of incorrect stabilizer outcomes of length l

could be misdiagnosed as an error of size $O(l^2)$ (the area of the loop). We don't anticipate this causing problems for the decoder as long as $l \ll L$, where L is the linear size of the lattice. We expect the probability of a length l loop of incorrect stabilizer outcomes to be suppressed in the size of the loop, so we can ensure that problematic loops are unlikely by increasing L. This relationship between faulty stabilizer outcomes and the errors that could have caused them is essentially "good soundness", a property that is a prerequisite for single-shot error correction using metachecks [51].

To test our intuition regarding resilience to measurement errors, we evaluated the performance of the SweepRule decoder for an error model that incorporates unreliable stabilizer measurements. We used an error model where each physical qubit of the code experiences a bit-flip error with probability p and stabilizer measurement outcomes are also flipped with probability p. To evaluate the effectiveness of the SweepRule decoder against this noise model, we estimate the error threshold of the decoder as a function of the number of error correction cycles, where we define an error correction cycle to be the following procedure:

1. Apply an X-error to each physical qubit independently with probability p.
2. Measure the Z-stabilizer generators perfectly to obtain the error syndrome.
3. Flip each stabilizer measurement outcome independently with probability p.
4. Apply the Sweep Rule simultaneously to each vertex of the lattice.

We simulate N error correction cycles, where can choose the sweep direction for each cycle. To understand how well the decoder is suppressing errors during this procedure, after the final cycle, we simulate reading out the information stored in the surface code. As we discussed in Section 1.3.1, measurement errors during readout have the same effect as physical qubit errors just prior to readout. Therefore, to simulate readout, we apply the bit-flip channel independently to each physical qubit and then we compute the values of the stabilizer generators. We then apply the Sweep Rule $32 \times L$ times, where L is the linear size of the surface code lattice. Following this, if the state of the surface code is not in the codespace (the $+1$ eigenspace of the stabilizer group) we record a failure, and otherwise we check whether a logical operator was applied during the procedure. We refer to the two phases of the decoding procedure as the error-suppression phase and the readout phase.

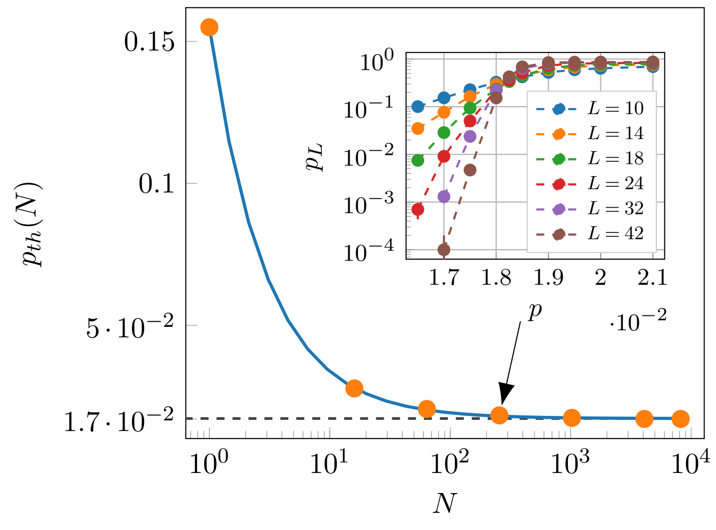
Using the procedure detailed above, we estimated the error threshold as a function of N (the number of error correction cycles) for cubic surface codes and rhombic-dodecahedral surface codes. We expect the error threshold to decay to a constant value, the sustainable threshold p_{sus} [102]. We used the following fitting function for both code families:

$$p_{th}(N) = p_{sus}\left(1 - \left(1 - \frac{p_{th}(1)}{p_{sus}}\right)N^{-\gamma}\right), \tag{3.35}$$

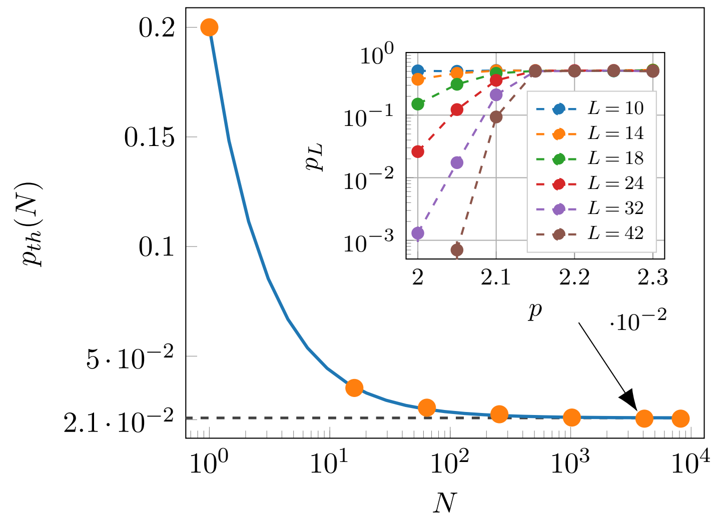
where γ is a parameter of the fit. We note that $p_{th}(1)$ is equal to the error threshold when measurements are perfect. We observe a sustainable threshold of $\approx 1.70\%$ for cubic surface codes (Figure 3.11a) and a sustainable threshold of $\approx 2.15\%$ for rhombic-dodecahedral surface codes (Figure 3.11b).

There are a number of parameters that we optimized to improve the performance of the decoder. The most important parameter is how often we change sweep direction. During the error-suppression phase, we found that changing the sweep direction after every $\approx \sqrt{L}$ applications of the rule was the best strategy, as shown in Figure 3.12. During the readout phase, we changed sweep direction after every L applications of the Sweep Rule. The frequency with which we change sweep direction is different for the error suppression and the readout phase because the priority for the decoder in each phase is different. In the error-suppression phase, our aim is to keep the spread of errors under control. Therefore, we don't want to sweep in one direction for too long because then errors will build up in the parts of the lattice that are problematic for this sweep direction (see Figure 3.4). On the other hand, in the readout phase there are no additional errors during the application of the rule, so our priority is to remove errors as fast as possible.

In Table 3.1, we compare the error thresholds for cubic surface codes (with boundaries) that we observed using the SweepRule decoder with error thresholds found using other decoders. We improve on the Toom's rule decoder used in [44] but our threshold value is lower than the threshold of a renormalization group (RG) decoder developed by Duivenvoorden et al. [42]. However, as we mentioned in Section 3.2, cellular automaton decoders are simple and parallelizable, which can be



a)



b)

Figure 3.11: Plots showing the sustainable threshold of cubic surface codes (Figure 3.11a) and rhombic-dodecahedral surface codes (Figure 3.11b). In each subfigure, we plot the error threshold, p_{th}, as a function of the number of error correction cycles, N. An error correction cycle consists of a bit-flip channel applied independently to every physical qubit, followed by an noisy measurement of the stabilizer generators, and finished with an application of the Sweep Rule simultaneously to all the vertices in the lattice. In both cases, the error threshold decays to a constant value, the sustainable threshold (dashed grey lines). We observe a sustainable threshold of $\approx 1.70\%$ for cubic surface codes (Figure 3.11a) and a sustainable threshold of $\approx 2.15\%$ for rhombic-dodecahedral surface codes. The blue line is the fit given in Equation 3.35, which shows excellent agreement with the data. The inset plots give examples of how we estimated the threshold for each data point in the main plots. In each inset, we plot the probability of a logical error, p_L, as a function of the qubit and measurement error probability, p, for lattices of growing linear size, L. The error threshold is the value of p where these curves intersect. The data in this Figure were generated using $\sim 187{,}000$ CPU hours. We acknowledge use of UCL supercomputing facilities (`https://www.ucl.ac.uk/research-it-services/services/research-computing-platforms`).

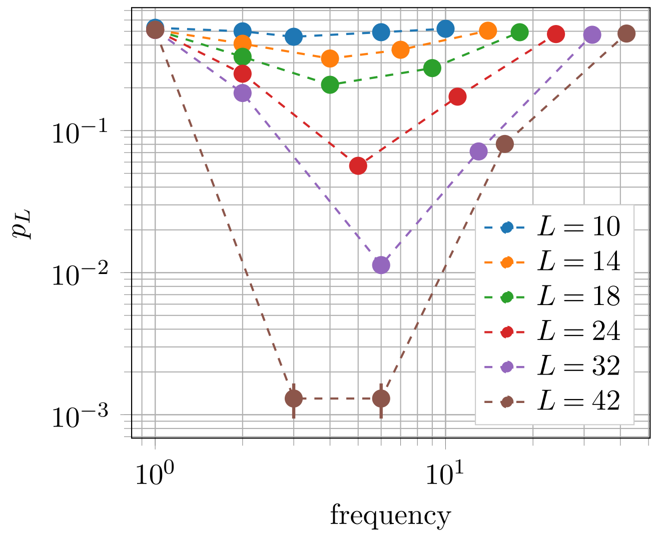


Figure 3.12: A plot showing the effect of varying the frequency with which we change the sweep direction during the error-suppression phase of decoding. We fix the number of error correction cycles to be $N = 1024$, and we choose an error probability $p = 0.021$ that is marginally below the error threshold. Then, for rhombic-dodecahedral lattices with different linear sizes, L, we estimate the logical error probability for different values of the frequency with which we change the sweep direction. We choose frequency values that are (approximately) evenly-spaced on a log scale. We achieve the best performance using frequencies $\approx \sqrt{L}$. We observe analogous behaviour in cubic surface codes.

Decoder	Perfect measurements	Unreliable measurements
Renormalization [42]	17.2%	7.3%
Toom's Rule [44]	12%	N/A
SweepRule	15.5%	1.7%

Table 3.1: Comparison of the error threshold of cubic surface codes against bit-flip noise for different decoders.

an advantage when compared to decoding algorithms that require more involved processing such as RG decoders. In addition, to deal with measurement errors, Duivenvoorden et al. repeat the stabilizer measurements $O(d)$ times before processing the syndrome history to find a correction, which further complicates their decoding procedure. We are not aware of any other error threshold results for rhombic-dodecahedral surface codes, but we anticipate that an RG decoder could be applied successfully to these codes.

Conclusion

In this Chapter, we have shown that 3D surface codes can effectively protect quantum information. However, the protection these codes offer is not symmetric as they are better at protecting against X-errors (loop-like syndromes) than against Z-errors (point-like syndromes). We showed that Z-errors can be decoded using the minimum-weight matching algorithm, but that multiple rounds of stabilizer measurement are required to deal with measurement errors. In contrast, X-errors in 3D surface codes can be corrected without repeating stabilizer measurements using our SweepRule decoder. The error thresholds we observed are also higher for X-errors than for Z-errors. However, we emphasize that the error thresholds in this chapter should be treated as approximate values. To get a better estimate of the error threshold, we would need to do a more detailed simulation where we take the circuits used to measure the stabilizers into account.

We note that the structure of the two types of error is an artefact of our surface code definition. If we exchange the definitions of the stabilizers, then X-errors would have point-like syndromes and Z-errors would have loop-like syndromes. Therefore, 3D surface codes always offer asymmetric protection, but we have the freedom to chose which type of error is better protected. Given this fact, surface codes may be well-suited to qubit technologies where the noise is biased towards either bit-flips or phase-flips e.g. [158, 159, 160].

It should be possible to improve on the decoders we have presented in this Chapter. In the case of Z-errors, as we mentioned in Section 3.1, there are other 2D surface code decoders that we could use in 3D codes if we want to prioritize speed. For example, the Union-Find decoder has almost linear runtime [146]. Alternatively, if we can construct a subsystem version of the 3D surface code in analogy to the 3D gauge colour code, then we may be able to correct Z-errors in a single shot i.e. without needing to repeat stabilizer measurements.

The SweepRule decoder we described in Section 3.2 achieves single-shot error correction of X-errors, but it has a lower threshold than an RG decoder (see Table 3.1). One possible way to improve the performance of our decoder is to apply the Sweep Rule multiple times between stabilizer measurements. If measurement is slow compared to classical computation in a particular qubit technology (e.g. trapped-ion qubits [161]), then the speed of the SweepRule decoder makes this strategy a possibility. We expect there would be some kind of trade-off between the measurement error probability and the optimal number of applications of the rule. We plan to investigate this issue in future work. It would also be interesting to test the performance of the SweepRule decoder against mode realistic noise models e.g. locally correlated errors. We expect that the decoder would still have a non-zero error threshold for such an error model, but the value of the threshold may be smaller than the threshold for independent bit-flip noise.

Finally, we expect the SweepRule decoder to be useful for other topological codes. For example, Kubica and Delfosse recently showed that we can use surface code decoders to decode colour codes [110]. Therefore, the SweepRule decoder could be used to decode X-errors in 3D colour codes with boundaries. It would also be interesting to see if we can extend the SweepRule decoder to hyperbolic surface codes. These codes are not locally Euclidean, which implies that our threshold proof does not work in this case. However, there is some evidence that local decoders perform well in such codes [162, Section 6.4], so it is worth investigating the performance of the SweepRule decoder in this context.

Chapter 4

Fault-tolerant three-dimensional surface code architectures

We can break down the problem of building a fault-tolerant quantum computer into smaller problems. These problems are: reliably preparing encoded $|0\rangle$ states, correcting errors that occur during the operation of the computer, implementing a universal gate set fault-tolerantly and accurately measuring the encoded qubits in the Z-basis. In this chapter, we combine the results of Chapters 2 and 3 to show that we can accomplish these tasks using 3D surface codes. In Section 4.1, we show how to implement the basic operations required in a quantum computer using a single stack of surface codes. Then, in Section 4.2, we generalize the techniques of lattice surgery [67] to 3D surface codes, showing how to transfer logical qubits between codes in different stacks. Finally, in Section 4.3, we propose two fault-tolerant quantum computing architectures based on 3D surface codes. After detailing these architectures, we estimate their resource requirements and compare these estimates with the requirements of the leading topological-code architectures. Much of the material in this chapter originally appeared in [112], and was carried out in collaboration with Dan Browne.

4.1 Fault-tolerance in a single stack

In this section, we explain how to do all of the basic operations required in a quantum computer using 3D surface codes. The basic unit we consider is a stack of three 3D surface codes defined on the same rectified cubic lattice. We defined a family of codes with this structure in Section 2.1.3, and we reproduce an image showing showing such a lattice in Figure 4.1.

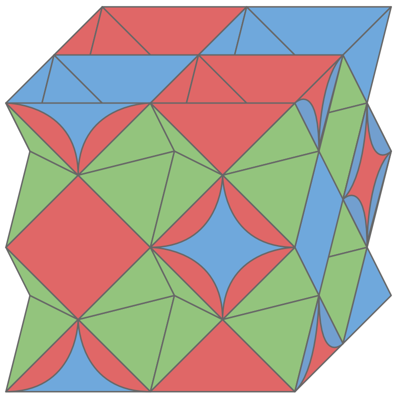


Figure 4.1: A rectified cubic lattice that supports three surface codes. We recall that each code, $\mathcal{SC}_c$, is indexed by the colour of its X-stabilizer generators, where $c \in \{r, g, b\}$. The Z-stabilizer generators of $\mathcal{SC}_c$ are associated with $c'c''$-faces, where $c' \neq c'' \neq c$. Each code in the stack has one logical qubit. Figure adapted from Vasmer and Browne [112].

First, we consider fault-tolerant error correction. In Chapter 3, we described decoders for correcting Z-errors (MWPM) and X-errors (SweepRule decoder) in 3D surface codes. We can correct each type of error independently because surface codes are CSS codes. In the case of Z-errors, we dealt with measurement errors by measuring the stabilizer generators $O(d)$ times before applying a correction, where d is the code distance. However, this is not necessary for X-errors. We showed that one can apply the SweepRule decoder even when measurements are unreliable and still observe an error threshold. Another approach to decoding X-errors when measurements are unreliable is to use the metacheck structure of the code. Metachecks are relations between the stabilizer generators, which allow us to tell if an observed syndrome contains measurement errors [51]. For example, in 3D surface codes, we know that valid X-error syndromes are loops of edges that are either closed or terminate on the boundaries. Therefore, it is possible to repair invalid error syndromes that contain broken loops (e.g. using MWPM), before applying a decoder. This is the approach that was taken in [42] for decoding errors in 4D surface codes, which is a similar problem to decoding X errors in 3D surface codes.

The second basic operation we consider is the preparation of encoded $|0\rangle$ states. We can generalize the state preparation technique used in 2D surface codes [28]

(Section 1.3.1) to 3D surface codes. We briefly review this technique here. To prepare an encoded $|0\rangle$ state fault-tolerantly, we prepare each of the physical qubits in the $|0\rangle$ state, before performing d rounds of error correction (where d is the code distance). In the case of the Z-errors, d rounds of error correction means measuring the X-stabilizer generators d times before applying a correction computed using the syndrome history. And for X-errors, d rounds of error correction means measuring the Z-stabilizer generators d times and applying the Sweep Rule simultaneously to each vertex after each measurement. This entire procedure takes $O(d)$ time.

In contrast, we can fault-tolerantly prepare $|+\rangle$ states in $O(1)$ time because X-errors in 3D surface codes can be corrected in a single shot. The procedure we use is essentially the state preparation procedure described by Dennis et al. for perfect stabilizer measurements [28]. We begin by preparing all the physical qubits in the $|+\rangle$ state, before measuring the Z-stabilizer generators. We can then use the metacheck structure of code to repair the syndrome, before applying a correction generated by a decoder. During this procedure, we may apply a logical X-operator, but this does not matter because the physical qubits started in a $+1$ eigenstate of $\overline{X}$, and measuring the Z-stabilizers will not change this eigenvalue. We note that it will not be possible to repair the syndrome perfectly, so there will be some residual errors caused by imperfect measurements. However, we can make these errors small as long as the measurement-error probability is smaller than the error threshold of the syndrome-repair procedure. Given that repairing a faulty syndrome is essentially a matching problem, it has been argued that the error threshold of this procedure will be similar to the error threshold of decoding Z-errors in 3D surface codes [42], which we estimated to be $\approx 2.9\%$ (Figure 3.1).

The third basic operation required of a fault-tolerant quantum computer is fault-tolerant measurement in the Z-basis. To do this, we generalize the fault-tolerant measurement procedure used in 2D surface codes [28] to 3D surface codes. Namely, to measure an encoded qubit in the Z-basis we simply measure all the physical qubits in the Z-basis and compute the eigenvalues of the Z-stabilizers. We then correct any X-errors implied by this syndrome (in practice, this means flipping measurement results). Finally, we compute the value of a $\overline{Z}$-operator using the corrected measurement results. To measure an encoded qubit in the X-basis we do the same procedure as explained above, except with X and Z interchanged. Both these measurement procedures take $O(1)$ time.

In Chapter 2, we showed how to implement a universal gate set in 3D surface codes without using magic state distillation (MSD). We proved that the CCZ gate is transversal for stacks of codes that are defined on rectified cubic lattices. We also showed that the CZ gate is transversal in such codes, and we presented a simple circuit for implementing a Hadamard gate that requires a single ancilla. We show the same circuit again in Figure 4.2. We can use this circuit to implement a single-qubit H-gate in a stack of three surface codes and to transfer a logical qubit between different qubits in the same stack. We denote the circuit in Figure 4.2 by $H_{cc'}$. This circuit takes the state $|\psi\rangle_c$ to the state $H|\psi\rangle_{c'}$, where the c and c' subscripts label the logical qubits of $\mathcal{SC}_c$ and $\mathcal{SC}_{c'}$, respectively. Consider the initial state $|\psi\rangle_r |+\rangle_g |+\rangle_b$. We can use sequences of $H_{cc'}$ circuits to transfer the state $|\psi\rangle_r$ from one code to another or to perform a single-qubit H as follows:

$$\begin{gathered} |\psi\rangle_r \xrightarrow{H_{rg}} H|\psi\rangle_g \xrightarrow{H_{gb}} |\psi\rangle_b, \\ |\psi\rangle_r \xrightarrow{H_{rg}} H|\psi\rangle_g \xrightarrow{H_{gb}} |\psi\rangle_b \xrightarrow{H_{br}} H|\psi\rangle_r. \end{gathered} \tag{4.1}$$

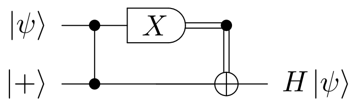


Figure 4.2: A circuit that implements a H-gate using an ancilla prepared in the $|+\rangle$ state, an X-basis measurement and CZ [78].

We have shown how to use 3D surface codes to implement the basic operations that are required in a fault-tolerant quantum computer. However, in a full quantum computing architecture, we need to be able to transfer logical qubits between surface codes in different stacks (or between 3D surface codes and 2D surface codes). We address this problem in the next section, where we generalize the techniques of lattice surgery to 3D surface codes.

4.2 3D surface code lattice surgery

Lattice surgery is a code deformation technique that allows us to merge two surface codes into a larger surface code or to split a surface code into two smaller surface codes. It was introduced by Horsman et al. as a way to encode logical qubits in a large sheet of 2D surface code [67]. Lattice-surgery merges and splits can be used to transfer qubits between codes or to implement $CNOT$ gates. There are two types of lattice surgery we can do in 3D surface codes: X-type and Z-type (corresponding to rough and smooth lattice surgery in the language of [67]). We start by presenting lattice surgery techniques for pairs of 3D surface codes before presenting a method for doing lattice surgery on a 3D surface code and a 2D surface code.

4.2.1 3D/3D lattice surgery

We start with X-type lattice surgery. Consider two rectified cubic lattices with boundaries (e.g. the lattice in Figure 4.1). Each lattice supports three surface codes, $\mathcal{SC}_c^{(i)}$, where $c \in \{r,g,b\}$ and $i \in \{1,2\}$ indexes the two stacks. We can do an X-type lattice-surgery merge between $\mathcal{SC}_c^{(1)}$ and $\mathcal{SC}_c^{(2)}$ by first aligning the c-boundaries of the two stacks (we recall that a c-boundary is a rough boundary in $\mathcal{SC}_c$ and a smooth boundary in the other codes). We complete the merge by preparing a layer of ancillas in the $|0\rangle$ state between the stacks and then measuring new X-stabilizers that join the two lattices. This configuration of lattices is shown in Figure 4.3. The product of the new X-stabilizers is $X_c^{(1)} \otimes X_c^{(2)}$, where $X_c^{(i)}$ is the logical operator of the code $\mathcal{SC}_c$ in stack i. There may also be new Z-stabilizers, which we add to the stabilizer group and measure in subsequent rounds of error correction. In addition, some Z-stabilizers on the boundaries where the merge took place may need to be modified. The merge operation maps $|\psi\rangle_c \otimes |\phi\rangle_c \to \alpha |\psi\rangle_c + (-1)^m \beta X |\psi\rangle_c$, where m is the outcome of the $X_c^{(1)} \otimes X_c^{(2)}$ measurement and $|\phi\rangle_c = \alpha |0\rangle_c + \beta |1\rangle_c$ [67].

To make the above procedure fault-tolerant, we need to do $O(d)$ rounds of error correction. This allows us to be confident about the value of $X_c^{(1)} \otimes X_c^{(2)}$. We note that any logical X-operator for either of the two initial codes is a valid logical X-operator for the merged code. However, to form a logical Z-operator in the new code we must join logical Z-operators from each of the initial codes into a single string of Z-operators that starts and ends at opposite c-boundaries. We can implement an X-type lattice-surgery split by measuring all the qubits in the layer where we want to split the lattice in the Z-basis. This splits the single surface code into two smaller surface codes. An X-type split performed on $\mathcal{SC}_c$ implements the following mapping: $\alpha' |+\rangle_c + \beta' |-\rangle_c \to \alpha' |++\rangle_c + \beta' |--\rangle_c$ [67].

Z-type lattice surgery is analogous to X-type lattice surgery. To perform a

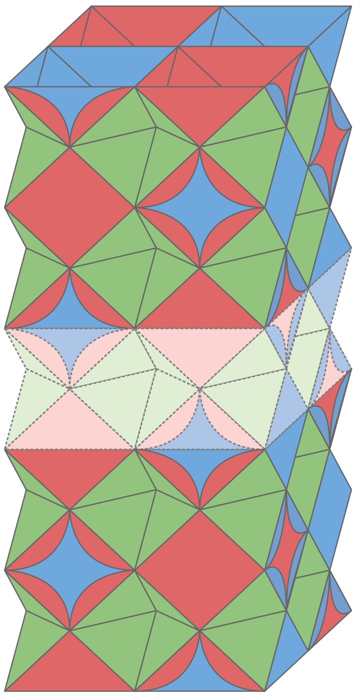


Figure 4.3: Lattice surgery in 3D surface codes. Both of the initial stacks (non-faded lattices) contain three surface codes. We prepare a layer of ancilla qubits (vertices of the faded lattice) and measure new stabilizers (faces and cells of the faded lattice) to merge codes of the same colour in separate stacks. To undo a merge, we simply measure the layer of ancilla qubits in the relevant basis. In this configuration, we can do X-type lattice surgery on $\mathcal{SC}_g^{(1)}$ and $\mathcal{SC}_g^{(2)}$, Z-type lattice surgery on $\mathcal{SC}_b^{(1)}$ and $\mathcal{SC}_b^{(2)}$, and Z-type lattice surgery on $\mathcal{SC}_r^{(1)}$ and $\mathcal{SC}_r^{(2)}$. Figure adapted from Vasmer and Browne [112].

Z-type merge on $\mathcal{SC}_c^{(2)}$ and $\mathcal{SC}_c^{(2)}$, we first align a c'-boundary of one stack with a c'-boundary of the other (this aligns the smooth boundaries of the codes). We then add a layer of ancilla qubits (all in the $|+\rangle$ state) and measure new Z-stabilizers that join the two lattices. There may also be new X-stabilizers and modified X-stabilizers at the join. The new Z-stabilizers (redundantly) tell us the value of $Z_c^{(1)} \otimes Z_c^{(2)}$. Unlike X-type lattice surgery, we can do Z-type lattice surgery in $O(1)$ time, as we can use the MWPM syndrome-repair procedure to get an accurate estimate of $Z_c^{(1)} \otimes Z_c^{(2)}$ from a constant number of stabilizer measurements. This illustrates that codes with single-shot error correction also have advantages when it comes to lattice surgery. Any logical Z-operator of either original code is a valid logical Z-operator of the merged code. However, the logical X-operators of the merged code are membranes of X-operators with boundaries that span the c' and c''-boundaries of the merged lattice. We can implement a Z-type split by measuring a layer of $\mathcal{SC}_c$ qubits in the X-basis. Figure 4.3 shows an example of Z-type lattice surgery on two 3D surface codes.

We note that we can simultaneously implement an X-type merge on the $\mathcal{SC}_c$ codes in different stacks, a Z-type merge on the $\mathcal{SC}_{c'}$ codes in different stacks and a Z-type merge on the $\mathcal{SC}_{c''}$ codes in different stacks. To do this we prepare a layer of qubits between c-boundaries of the two stacks we want to merge. At every vertex in the new layer we place three qubits (one for each pair of codes), prepared in the state $|0\rangle_c |+\rangle_{c'} |+\rangle_{c''}$. We then modify the stabilizer groups of all three pairs of codes at once as discussed in the previous paragraphs to merge the three pairs of codes simultaneously. We can also invert this process to do a simultaneous split on all three pairs of codes.

4.2.2 2D/3D lattice surgery

We can do Z-type lattice surgery on a 2D surface code and a 3D surface code using procedures that are similar to 2D surface code lattice surgery. However, performing X-type lattice surgery on a 2D surface code and a 3D surface code is more complicated. This is because the dimension of the logical Z-operators in 2D surface codes and 3D surface codes is the same whereas the dimension of the logical X-operators is not. Therefore, we only discuss Z-type lattice surgery in this section.

We start with a 3D surface code stack and a 2D surface code sheet aligned such that the 2D sheet is in the same plane as the bottom layer of the 3D stack (see Figure 4.4). To do a lattice-surgery merge, we simply measure new Z-stabilizers whose product is $Z_{2D} \otimes Z_{3D}$ (the tensor product of the logical Z-operators of the codes). The X-stabilizers of both codes at the join will also be modified. Because single-shot error correction is not possible with 2D surface codes, we need to do $O(d)$ rounds of error correction to ensure fault tolerance, where d is the code distance. The effect of the Z-type merge on the logical operators is more interesting in the 2D/3D case than the 3D/3D case. The logical Z-operators of the original codes are valid logical Z-operators of the merged code. However, logical X-operators of the merged code are products of membrane operators in the 3D lattice and string operators in the 2D lattice. This structure means that when decoding X-errors in

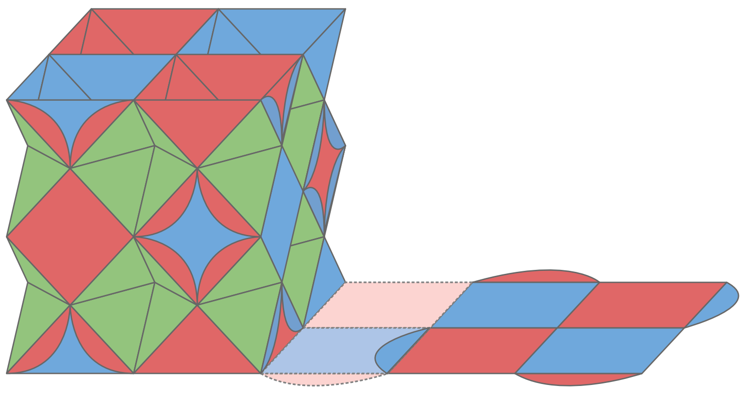


Figure 4.4: Z-type lattice surgery on 3D and 2D surface codes. We associate X-stabilizer generators with b-faces and Z-stabilizer generators with r-faces in the 2D surface code. In the stack we consider $\mathcal{SC}_b$ (X-stabilizer generators associated with b-cells and Z-stabilizer generators associated with rg-faces). The left and right boundaries of the 2D surface code are smooth boundaries and the left and right boundaries of the stack are r-boundaries (smooth boundaries in $\mathcal{SC}_b$). To implement a lattice-surgery merge between the two codes we measure two new Z-stabilizers (faded r-faces), whose product is $Z_{2D} \otimes Z_{3D}$. We also merge the weight-two X-stabilizer on the left boundary of the 2D code with the weight-three X-stabilizer associated with the bottom r-face on the right boundary of the 3D code. This stabilizer is represented by the faded blue face in the Figure. To undo the merge operation we return to measuring the pre-merge stabilizers. Figure adapted from Vasmer and Browne [112].

the combined 2D and 3D codes, we would have to use a different decoder in each code and combine their results. To implement a Z-type split we simply return to measuring the pre-merge stabilizers. Finally, we note that the Z-type lattice surgery operations we have described can also be used to do Z-type lattice surgery between two 3D surface codes.

As we previously stated, we can use lattice surgery to implement $CNOT$ gates and to transfer qubits between different surface codes. This is because lattice surgery allows us to measure products of encoded Pauli operators fault-tolerantly [67, 68]. A Z-type lattice-surgery merge followed by a Z-type lattice surgery split is equivalent to a logical $Z \otimes Z$ measurement. Figure 4.5 shows a circuit that uses a $Z \otimes Z$ measurement to transfer a qubit from one surface code to another. We can also measure $X \otimes X$ fault-tolerantly by implementing an X-type lattice surgery merge followed by a split. Measuring in both the X and Z-basis enables us to implement a $CNOT$ gate [67, 68], as shown in Figure 4.6. We can also use lattice surgery to

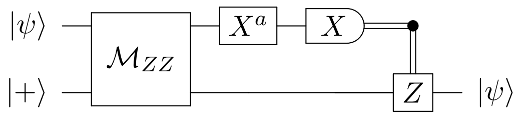


Figure 4.5: A circuit that transfers a qubit $|\psi\rangle$ to an ancilla prepared in the $|+\rangle$ state using a $Z\otimes Z$ measurement, which is denoted above by $\mathcal{M}_{ZZ}$. The binary variable $a=(1-m)/2$, where $m\in\{-1,1\}$ is the outcome of the $Z\otimes Z$ measurement.

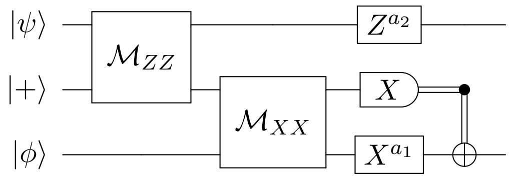


Figure 4.6: A circuit that implements a $CNOT$ gate with $|\psi\rangle$ as the control qubit and $|\phi\rangle$ as the target qubit. $\mathcal{M}_{ZZ}$ denotes a $Z\otimes Z$ measurement and $\mathcal{M}_{XX}$ denotes a $X\otimes X$ measurement. The binary variables $a_1=(1-m_1)/2$ and $a_2=(1-m_2)/2$, where $m_1\in\{-1,1\}$ and $m_2\in\{-1,1\}$ are the outcomes of the $Z\otimes Z$ and $X\otimes X$ measurements, respectively.

perform more complex (Clifford) gates [163, 68].

4.3 Architectures and overheads

As we discussed in Chapter 1, the large resource cost of MSD has motivated research into alternative implementations of non-Clifford gates in topological codes. To reason about the resource requirements of different architectures, we use a space-time overhead metric. We say that an architecture that requires n physical qubits per logical qubit and d rounds of stabilizer measurement per operation has a space-time overhead of nd. We neglect the time cost of efficient classical computation, as we are primarily interested in the quantum computing resources. Using the space-time overhead metric allows us to compare the scaling of the resource costs of different architectures without descending into highly detailed analyses of each architecture.

A 2D surface code architecture (e.g [15, 67, 70, 69]) using distance d codes requires $\approx d^2$ physical qubits per logical qubit and $O(d)$ rounds of stabilizer measurement per operation. Therefore, the total space-time overhead of this architecture is $O(d^3)$. The 3D gauge colour code architecture proposed by Bombín [101] requires $\approx d^3$ physical qubits per logical qubit, but only needs a constant number of rounds of stabilizer measurement per operation. So this architecture also has a space-time overhead of $O(d^3)$. Therefore, using the space-time overhead metric, a 2D surface code architecture and a 3D gauge colour code architecture have comparable resource costs. Clearly, the locality of the native gates (i.e. whether 3D connectivity is possible) in the physical system used to realize the qubits has a significant impact on which architecture is preferable. But overall, given current qubit error rates [103, 104], the 2D surface code's superior error threshold gives it an advantage over the gauge colour code. However, we note that 2D surface code architectures have been heavily optimized over the past seventeen years whereas the gauge colour code was only introduced relatively recently. Therefore, it seems unwise to dismiss gauge colour code architectures at this stage.

In the remainder of this section, we propose two architectures that utilize 3D surface codes and compare them with the architectures we discussed above. To begin, we present an architecture that exclusively uses 3D surface codes to encode logical qubits. And in our second proposal, we suggest using 3D surface codes to produce magic states, before transferring these states into a 2D surface code architecture where they can be used to implement non-Clifford gates.

4.3.1 Purely 3D architecture

Here, we present a fault-tolerant quantum computing architecture where every qubit is encoded in a 3D surface code. The fact that we use 3D surface codes makes our architecture unsuitable for qubit technologies with planarity constraints e.g. superconducting qubits fabricated on 2D chips [103]. However, in other architectures do not have the same constraints. For example, all-to-all coupling can be engineered in ion trap qubits [104]. Our proposal seems like it may be well suited to a networked architecture, where cells containing matter qubits are connected by photonic links [164, 165, 166, 167, 168, 169]. Alternatively, 3D codes are well suited to ballistic linear optical architectures, because in such architectures the third dimension becomes the time dimension [170, 64, 171, 172, 173]. Finally, we recall from Section 3.2.3 that 3D surface codes are well-suited to systems where the noise is biased (e.g. [158, 159, 160]) because the protection offered by the codes is asymmetric. We emphasize that redefinition of the stabilizers allows us to choose which errors (X or Z) have loop-like syndromes (single-shot decoding, higher threshold) or point-like syndromes (multiple rounds of stabilizer measurement, lower threshold).

We now describe the structure of our architecture. Consider a large rectified cubic lattice which we divide into "stacks" such that every stack is a rectified cubic

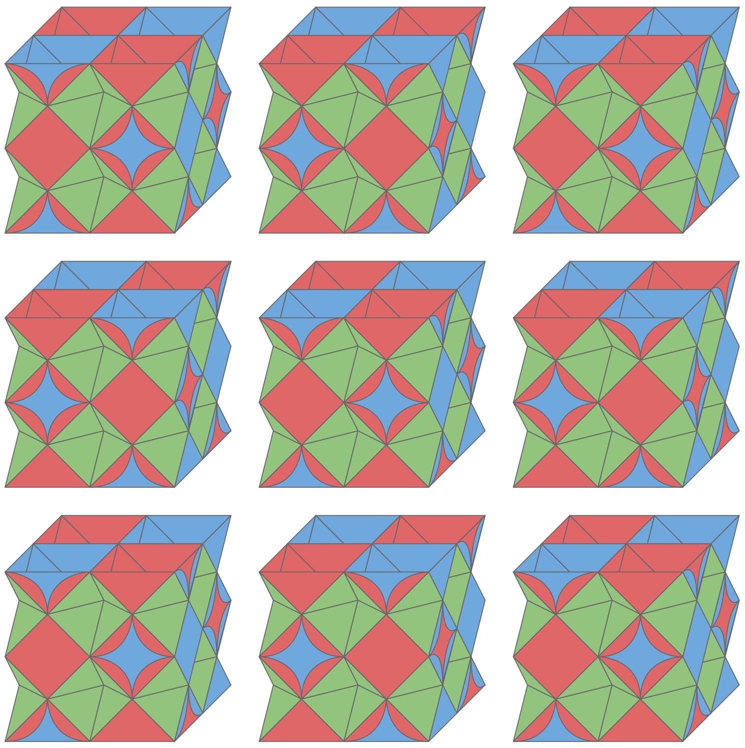


Figure 4.7: The macro-structure of our purely-3D surface code architecture. We divide a large rectified cubic lattice into equally-sized stacks such that every stack (in the bulk) is adjacent to six other stacks. In the Figure, we show the stacks in two out of the three dimensions of a $3 \times 3 \times 3$ layout of distance-3 codes.

lattice with boundaries (see Figure 4.7). Each stack therefore has three logical qubits and is adjacent to six other stacks. As we detailed in Section 4.1, we can do fault-tolerant error correction, state preparation and measurement in each stack. In addition, we can do lattice surgery on adjacent stacks. This allows us to transfer logical qubits between different stacks. We use half the stacks to store information (data qubits) and half as (encoded) ancillas, where the role of the stacks alternate. This means that we can use lattice surgery to implement a $CNOT$ gate between any two data qubits which are adjacent to the same ancilla stack. If the two data qubits are encoded in different colour surface codes, then we need to use two ancilla qubits, otherwise we can use a single ancilla. The second ancilla is needed because we can only do lattice surgery between codes of the same colour. $CNOT$ gates allow us to swap any two data qubits which are adjacent to the same ancilla stack.

As we showed in Section 4.1, we can implement the universal gate set

$\{CCZ, H, CZ\}$ in a single stack without using MSD. Our implementations of CCZ and CZ are transversal but, as we discussed in Section 4.1, to apply a H-gate to a data qubit we need to use the other two data qubits in the stack as ancillas. To avoid this, we implement the H-gate by the following method: first we transfer the data qubit to an adjacent ancilla stack, then we use the procedure in Equation 4.1 to apply the H-gate to the data qubit, before transferring it back to its original stack.

To summarize, in our architecture, we can swap arbitrary data qubits and implement a universal gate set in each stack. CCZ gates can be performed in parallel on all data qubits, CZ gates can be performed in parallel on two thirds of the data qubits and H-gates can be performed in parallel on a third of the data qubits. The CCZ gate, CZ gate and measurement can be done in $O(1)$ time steps (assuming fast classical computation), whereas the H-gate, $|0\rangle$ state preparation, and error correction all require $O(d)$ time steps. Each 3D surface code requires $\approx d^3$ physical qubits per logical qubit. Therefore, the space-time overhead of our architecture is $O(d^4)$, which is larger than the space-time overhead of 2D surface code architectures and the 3D gauge colour code architecture.

The reason for the larger space-time overhead of our architecture is that we require $O(d)$ rounds of stabilizer measurement for a subset of the operations we want to implement. The main reason for this is that repeated stabilizer measurements are required to correct Z-errors fault-tolerantly. Another disadvantage of our architecture is that the error threshold of the 3D surface code is likely to be smaller than the error threshold of the 2D surface code. The error threshold of the 3D surface code is limited by the Z-error threshold, which we estimated to be $\approx 1.25\%$. This estimate is almost certainly too optimistic because we did not take the stabilizer measurement circuits into account. We expect these circuits to introduce more errors, especially for high-weight stabilizers such as those found in tetrahedral-octahedral surface codes (see Figure 2.8). In future, we plan to investigate the possibility of constructing "gauge 3D surface codes" that share some of the properties of the gauge colour code. If such codes exist, it may be possible use them in an architecture with $O(d^3)$ space-time overhead and single-shot error correction which could be competitive with the pre-eminent topological code architectures.

4.3.2 Hybrid 2D/3D architecture

The basic idea of our hybrid architecture is to use 3D surface codes to realize a non-Clifford gate in a 2D surface code architecture, thereby eliminating the necessity of doing MSD. We consider 2D surface code architectures where logical qubits are encoded in patches of surface code and lattice surgery is used to couple qubits in different patches e.g. [67, 68, 163, 70, 69]. The Clifford group can be implemented fault-tolerantly with low overhead in such an architecture [68, 66]. To promote a 2D lattice-surgery architecture to universality, we use 3D surface codes as CCZ state factories, where a CCZ state is defined to be $|CCZ\rangle = CCZ\,|{+}{+}{+}\rangle$. We can consume a CCZ state to apply a CCZ gate to three logical qubits using the Clifford circuit shown in Figure 4.8. Therefore, the architecture we have just described fulfils all the criteria of a fault-tolerant architecture, and the role of the 3D surface codes is restricted to the area where they are most useful.

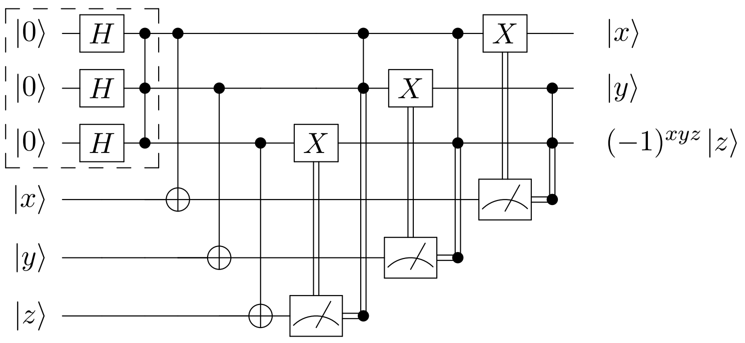


Figure 4.8: A circuit that consumes one CCZ state (dashed box) to implement a CCZ gate on the bottom three qubits. We note that $H_t \times CNOT_{ct} \times H_t = CZ$, where c and t refer to the control and target qubits. We constructed this circuit using the methodology described in [78].

Let us consider the creation of CCZ states using 3D surface codes in more detail. The first step in the process is to prepare an encoded $|{+}{+}{+}\rangle$ state, which we can do fault-tolerantly in constant time, as we explained in Section 4.1. Next, we transversally apply the CCZ gate which takes a single time step. Therefore, producing CCZ states encoded in 3D surface codes requires $O(1)$ rounds of stabilizer measurement. We emphasize that the procedure for making CCZ states is simple, so we expect that 3D surface code error correction would be easier in this context and the value of the error threshold would be higher than in the purely 3D architecture described in Section 4.3.1.

After producing CCZ states, we need to transfer them from the 3D surface codes to the 2D surface code architectures. We can do this using lattice surgery (as explained in Section 4.2.2) which takes time $O(d)$. Alternatively, we can use a dimensional jump [101] to transform a 3D surface code into a 2D surface code in constant time. To perform such a jump, we measure all the physical qubits in the X-basis, apart from the qubits on one of the boundaries. We use the measurement outcomes to compute the value of the X-stabilizers, and we process this information to find a correction, which we apply to the qubits on the boundary. This procedure fault-tolerantly transforms a 3D surface code into a 2D surface code, as long as the boundary of the 3D code has the structure of a 2D code. More specifically, we require that if we restrict the stabilizer group of the 3D surface code to a boundary, then we recover the stabilizer group of a 2D surface code defined on that boundary. The codes in our family of stacked codes have this property, as we explain in Figure 4.9.

It is not possible to perform a dimensional jump to the same boundary for all three codes in the stack. This may be a problem if the 2D surface codes in our architecture all lie in the same plane as one of the boundaries of the 3D surface code. For example, suppose the 2D surface code patches lie in the same plane as a 3D surface code boundary that supports dimensional jumps for $\mathcal{SC}_g$ and $\mathcal{SC}_r$ (e.g. the boundary highlighted in Figure 4.9). To move a CCZ state from the 3D surface code stack to the 2D surface code part of the architecture, we would first transfer the $\mathcal{SC}_g$ and $\mathcal{SC}_r$ parts of the state via dimensional jumps. Next, we would prepare new encoded $|+\rangle$ state ancillas in $\mathcal{SC}_r$ and $\mathcal{SC}_g$, and use the circuit in Figure 4.2 to transfer the $\mathcal{SC}_b$ part of the CCZ state to $\mathcal{SC}_r$. Finally, we would use a dimensional jump to finish transferring the CCZ state to the 2D surface code part of the architecture.

In the architecture we have just described we need $\approx d^2$ physical qubits per logical qubit and $O(d)$ rounds of stabilizer measurement for the 2D surface code part. And in the CCZ factories we require $O(d^3)$ physical qubits and $O(1)$ rounds of error correction to produce each CCZ state. The total resource requirements of our architecture are approximately $[c_2 O(d^2) + c_3 O(d^3)] \times [c_2 O(d) + c_3 O(1)]$, where c_2 and c_3 are the number of required 2D surface codes and 3D surface codes, respectively. If we assume that the required number of 3D surface codes is a factor of $\approx d$ smaller than the required number of 2D surface codes, then the overall resource re-

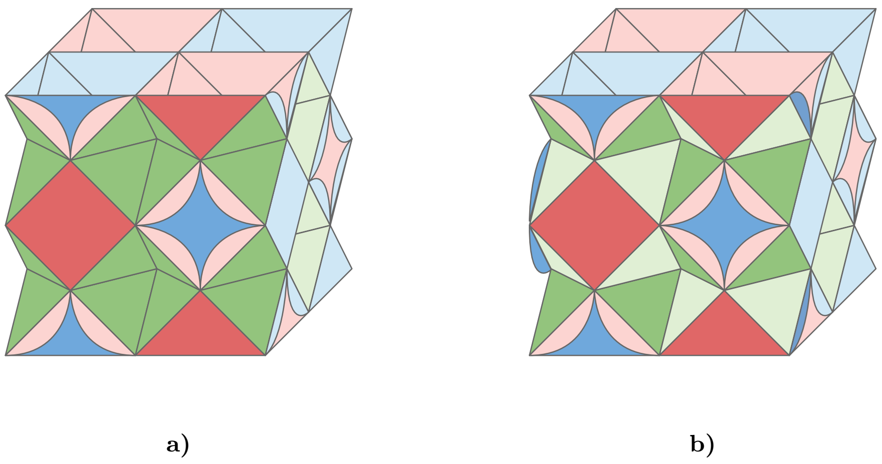


Figure 4.9: To perform a dimensional jump, we require that if we restrict the stabilizer group of a 3D surface code to a boundary, then we recover the stabilizer group of a 2D surface code defined on the boundary. Consider the front boundary of the lattice. In Figure 4.9a, we highlight a subset of the stabilizers of $\mathcal{SC}_g$ (non-faded faces). We recall that this code has X-stabilizer generators associated with g-cells and Z-stabilizer generators associated with rb-faces (faces shared between r-cells and b-cells). By comparing this figure with the 2D surface code shown in Figure 2.2, we see that the highlighted stabilizers are exactly the stabilizers of a [[13,1,3]] 2D surface code. In Figure 4.9b, we highlight a subset of the stabilizers of $\mathcal{SC}_r$. We recall that this code has X-stabilizer generators associated with r-cells and Z-stabilizer generators associated with bg-faces. In addition, we associate weight-four $\mathcal{SC}_r$ X-stabilizers with the b-faces on the front boundary. The highlighted X-stabilizers are exactly the stabilizers of a [[13,1,3]] 2D surface code. And by taking products of the highlighted Z-stabilizers, we can recover the Z-stabilizers of the same 2D surface code.

quirements of our architecture would be $O\left(d^3\right)$. This assumption is realistic because a 3D surface code magic-state factory would produce magic states extremely quickly which means that the spatial footprint of the factory would be small. The overhead of MSD is estimated to also scale like $O\left(d^3\right)$, but with a significant constant hidden by the Big-O notation [86]. Therefore, depending on the physical systems used to build the qubits, it may be beneficial to use 3D surface codes instead of MSD to implement a non-Clifford gate in a 2D surface code architecture. However, to conclusively answer the question of which approach is preferable, we would need to perform a much more detailed analysis of the resource costs of both approaches. We defer this analysis until future work.

We observe that it may be possible to combine 3D surface codes and MSD. For example, we could use small 3D surface codes to produce noisy CCZ states which could be fed into a MSD protocol (e.g. [72]). This may eliminate the need for the multiple rounds of MSD that can be required, depending on the target error rate. Finally, as we discussed in the previous section, the non-planarity of 3D surface codes means that they are most likely to be useful for qubit technologies where non-planar connectivity is relatively easy to engineer (e.g. ion-trap qubits [104] or photonic qubits [173]).

Conclusion

In this chapter, we proposed two quantum computing architectures based on 3D surface codes, and we evaluated their performance using a space-time metric. Out of the two, the hybrid 2D/3D surface code architecture is the most promising, as in this architecture we limit the role of the 3D surface codes to producing CCZ states, which can be done in constant time (ignoring the cost of classical computation). In addition, the error-correction problem is easier for 3D surface code CCZ-state factories than in an architecture where all operations must be implemented using 3D surface codes. We envisage using the CCZ states to implement the (non-Clifford) CCZ gate in a 2D surface code architecture, thus achieving universality without using MSD. Furthermore, in our hybrid architecture, we achieve the same space-time overhead as 2D surface code architectures that use MSD to achieve universality.

Brown recently proposed a method for doing a CCZ gate in a 2D surface code architecture that builds on the results in this thesis [174]. His method also relies on a recent proposal by Bombín, who showed that it is possible to realize a 3D colour code using only a constant thickness slice of 2D colour code [175]. Brown combined the transversal CCZ in 3D surface codes (Section 2.2) with Bombín's insight to propose an architecture which is similar to our hybrid architecture. Specifically, he considered a 2D surface code architecture where MSD has been replaced by a simulation of a 3D surface code CCZ state factory using 2D surface codes. Brown's proposal is attractive because it allows us to use the processing power of 3D surface codes in a 2D architecture. However, using a 2D surface code to simulate a 3D surface code makes the decoding problem significantly more difficult. At present, it is not clear what the error tolerance of Brown's scheme is. We plan to investigate this question in future work.

Chapter 5

Conclusion

Given current technology, there is no way of avoiding error-correction and fault-tolerance if we want to build a quantum computer capable of running large-scale quantum algorithms. However, the number of qubits required in the leading fault-tolerant architectures is many orders of magnitude larger than the number of qubits in the small quantum computers that are available today. Reducing the overhead of fault-tolerance is, therefore, a central problem in the field of quantum computing.

In this thesis, we examined the question of whether using 3D surface codes allows us to reduce the overhead of fault-tolerance. Firstly, we showed that 3D surface codes have a noteworthy property when it comes to processing information: they possess a transversal (non-Clifford) CCZ gate. Secondly, we studied the decoding problem in 3D surface codes. We observed that 3D surface codes offer asymmetric protection against errors. We adapted the SweepRule decoder to work for codes with boundaries, and we showed that one can use this decoder to decode bit-flip errors even if the error syndrome is unreliable. However, to decode phase-flip errors, we needed to use a relatively complex (but still efficient) classical algorithm to process the information from repeated stabilizer measurements. Thirdly, we proposed two quantum computing architectures that utilize 3D surface codes. We provided a method for implementing a universal and fault-tolerant gate-set in a single stack of 3D surface codes using transversal gates and Pauli measurements. And we generalized lattice surgery to 3D surface codes, which enabled us to envisage a quantum computing architecture where the basic building block is a stack of 3D surface codes. In addition, we showed that one can use a 3D surface code to produce magic states in a 2D surface code architecture (removing the need for MSD).

We evaluated the overhead of our architectures using a space-time metric and we compared these results with the overhead of two leading architectures: 2D surface code architectures and gauge colour code architectures. We found that our purely 3D architecture has a larger space-time overhead than the leading architectures. This is because, unlike gauge colour codes, single-shot error correction of both bit-flip and phase-flip errors is not possible in 3D surface codes. Despite this, we showed that using 3D surface codes to produce magic states in a 2D surface code architecture has a space-time overhead that is comparable to the leading architectures. This result relies on two non-trivial properties of 3D surface codes: the transversality of CCZ, and single-shot error correction of bit-flip errors. Consequently, the space-time overhead of producing magic states using 3D surface codes is $O\left(d^3\right)$, where d is the code distance. MSD protocols have been shown to have space-time requirements with the same scaling [86] and significant constant factors hidden by the Big-O notation. Therefore, there may be some parameter regimes where using 3D surface codes to produce magic states is more efficient than using MSD.

A number of research questions follow naturally from this work. Firstly, given the remarkable properties of the gauge colour code [49, 101, 50], it is logical to ask whether a “gauge surface code” can be constructed. In 2D, such a code exists [176], however, this code is in the same topological phase of the 2D surface code [176], i.e. there exists a geometrically-local, constant-depth circuit that transforms any 2D gauge surface code into a regular 2D surface code. This is in contrast to the gauge colour code, which is not in the same topological phase as the 3D colour code. In fact, understanding the properties of the gauge colour code Hamiltonian is still an active area of research [177, 178]. We would like to construct a 3D gauge surface code with low weight gauge operators and single-shot error correction of both bit-flip and phase-flip errors. This in turn would enable us to reduce the space-time cost of our 3D surface code architecture. A 3D gauge surface code may also be interesting from a condensed-matter perspective, and may be easier to analyse than the gauge colour code because surface codes generically have less complex topological excitations than colour codes.

In the conclusion of Chapter 4, we briefly discussed a recent proposal by Brown [174] for implementing a CCZ gate in a 2D surface code architecture. Brown’s work combines the results of Chapter 2 (published in [112]) with a recent proposal by Bombín [175]. Brown envisages using 3D surface codes to produce CCZ states, but with one of the spatial dimensions in a 3D surface code becoming the time dimension. Therefore, instead of building a full 3D surface code, it is only necessary to build a constant thickness slice of it. This slice can then be used to move through the 3D surface code lattice in time, implementing a transversal CCZ gate at each time step. It is natural to worry about the resilience of Brown's CCZ implementation to error, as we do not expect a family of constant thickness surface codes to have an error threshold. However, Brown's CCZ implementation is made fault-tolerant using a just-in-time (JIT) decoder (a concept also introduced by Bombín [175]), which can correct errors using only a subset of the error syndrome of the full 3D code. Brown proved that JIT decoding has an error threshold, but this value is very low ($p \sim 10^{-15}$), as is often the case with analytically-proven thresholds. At present, no numerical estimates of the error threshold have been undertaken. If Brown's scheme has a high error threshold then it may be more advantageous than using MSD, even for qubit technologies where 3D connectivity is problematic. Due to the fact that our SweepRule decoder only needs to process information locally, we conjecture that it would be suitable for JIT decoding. We plan to apply our decoder to this problem in future work.

Finally, we can ask a more general question about the fundamental space-time overhead of doing universal fault-tolerant quantum computation with topological stabilizer codes. All of the schemes we are aware of have at best a space-time overhead that scales like d^3, where d is the code distance. It is logical to ask whether this upper bound is also a lower bound. For 3D topological codes this is clearly the case, as the best we can hope for using d^3 qubits is operations that take constant time. However, the situation is less clear for 2D topological codes, where the space overhead is $\approx d^2$. It seems unlikely that we will be able to implement the required operations in constant time, because doing this in 3D codes requires single-shot error correction, which we know is not possible in 2D topological stabilizer codes [106, 105]. In future work, we plan to investigate the fundamental space-time overhead of universal computation with topological codes more thoroughly. Such an analysis should also allow us to compare more easily the overhead of the various topological code quantum computing architectures. This in turn would help us to answer the question of which architecture is best suited to particular qubit technologies.

Appendix A

Proof of Lemma 1

Lemma 1 was required in the proof of the transversality of CCZ for 3D surface codes. We restate it here:

Lemma 1. Given a finite set of binary vectors $\{a_j\}$, the parity of their sum is equal to the sum of their parities.

Proof. An equivalent statement of the Lemma 1 is that for any finite set of m binary vectors $\{a_j\}$, the following Equation holds:

$$\sum_{j=1}^{m} |a_j| - |\sum_{j=1}^{m} a_j| = 2t, \tag{A.1}$$

where t is a positive integer and $|a_j|$ denotes the Hamming weight of a_j. We prove Lemma 1 by induction. Consider the $m = 2$ case. We have

$$|a_1 + a_2| = |a_1| + |a_2| - 2\mathcal{O}(a_1, a_2), \tag{A.2}$$

where $\mathcal{O}(a,b)$ is the overlap of a and b *i.e.* the number of positions where both a and b are equal to one. Rearranging, we obtain

$$|a_1| + |a_2| - |a_1 + a_2| = 2\mathcal{O}(a_1, a_2). \tag{A.3}$$

Now consider the inductive step from m to $m+1$:

$$\begin{aligned}
|\sum_{j=1}^{m+1} a_j| &= |\sum_{j=1}^{m} a_j| + |a_{m+1}| - 2\mathcal{O}\left(\sum_{j=1}^{m} a_j, a_{m+1}\right) \\
&= \sum_{j=1}^{m} |a_j| - 2t + |a_{m+1}| - 2\mathcal{O}\left(\sum_{j=1}^{m} a_j, a_{m+1}\right) \\
&= \sum_{j=1}^{m+1} |a_j| - 2\left(t + \mathcal{O}\left(\sum_{j=1}^{m} a_j, a_{m+1}\right)\right).
\end{aligned} \tag{A.4}$$

□

Bibliography

[1] P. W. Shor. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM J. Comput.*, **26**, 1484 (1997).

[2] A. W. Harrow, A. Hassidim, and S. Lloyd. Quantum algorithm for linear systems of equations. *Phys. Rev. Lett.*, **103**, 150502 (2009).

[3] M. Reiher, N. Wiebe, K. M. Svore *et al.* Elucidating reaction mechanisms on quantum computers. *Proc. Natl. Acad. Sci.*, **114**, 7555 (2017).

[4] IBM announces advances to IBM Quantum systems & ecosystem. `https://www-03.ibm.com/press/us/en/pressrelease/53374.wss`. Accessed: 10-08-2019.

[5] A preview of Bristlecone, Google's new quantum processor. `https://ai.googleblog.com/2018/03/a-preview-of-bristlecone-googles-new.html`. Accessed: 10-08-2019.

[6] The Rigetti 128-qubit chip and what it means for quantum. `https://medium.com/rigetti/the-rigetti-128-qubit-chip-and-what-it-means-for-quantum-df757d1b71ea`. Accessed: 10-08-2019.

[7] C. Gidney and M. Ekerå. How to factor 2048 bit RSA integers in 8 hours using 20 million noisy qubits. *arXiv:1905.09749* (2019).

[8] A. Y. Kitaev. Fault-tolerant quantum computation by anyons. *Ann. Phys.*, **303**, 2 (2003).

[9] S. B. Bravyi and A. Y. Kitaev. Quantum codes on a lattice with boundary. *arXiv:quant-ph/9811052* (1998).

[10] M. H. Freedman and D. A. Meyer. Projective plane and planar quantum codes. *Found. Comput. Math.*, **1**, 325 (2001).

[11] R. Raussendorf and J. Harrington. Fault-tolerant quantum computation with high threshold in two dimensions. *Phys. Rev. Lett.*, **98**, 190504 (2007).

[12] A. G. Fowler, A. M. Stephens, and P. Groszkowski. High-threshold universal quantum computation on the surface code. *Phys. Rev. A*, **80**, 052312 (2009).

[13] D. S. Wang, A. G. Fowler, and L. C. L. Hollenberg. Surface code quantum computing with error rates over 1%. *Phys. Rev. A*, **83**, 020302 (2011).

[14] A. M. Stephens. Fault-tolerant thresholds for quantum error correction with the surface code. *Phys. Rev. A*, **89**, 022321 (2014).

[15] A. G. Fowler, M. Mariantoni, J. M. Martinis *et al.* Surface codes: Towards practical large-scale quantum computation. *Phys. Rev. A*, **86**, 032324 (2012).

[16] M. A. Nielsen and I. L. Chuang. *Quantum computation and quantum information: 10th anniversary edition.* Cambridge University Press, 10th edition (2011).

[17] K. Kraus. *States, effects and operations: fundamental notions of quantum theory.* Springer (1983).

[18] P. W. Shor. Scheme for reducing decoherence in quantum computer memory. *Phys. Rev. A*, **52**, R2493 (1995).

[19] A. M. Steane. Error correcting codes in quantum theory. *Phys. Rev. Lett.*, **77**, 793 (1996).

[20] L. Viola, E. Knill, and S. Lloyd. Dynamical decoupling of open quantum systems. *Phys. Rev. Lett.*, **82**, 2417 (1999).

[21] D. A. Lidar, I. L. Chuang, and K. B. Whaley. Decoherence-free subspaces for quantum computation. *Phys. Rev. Lett.*, **81**, 2594 (1998).

[22] D. Gottesman. *Stabilizer codes and quantum error correction.* Ph.D. thesis, Caltech (1997).

[23] A. R. Calderbank and P. W. Shor. Good quantum error-correcting codes exist. *Phys. Rev. A*, **54**, 1098 (1996).

[24] Z. Cai, M. A. Fogarty, S. Schaal *et al.* A silicon surface code architecture resilient against leakage errors. *arXiv:1904.10378* (2019).

[25] A. Steane. Multiple-particle interference and quantum error correction. *Proc. Royal Soc. Lond., Ser. A: Math. Phys. Eng. Sci.*, **452**, 2551 (1996).

[26] M. B. Hastings. Trivial low energy states for commuting hamiltonians, and the quantum PCP conjecture. *Quantum Inf. Comput.*, **13**, 393 (2013).

[27] N. Delfosse, P. Iyer, and D. Poulin. Generalized surface codes and packing of logical qubits. *arXiv:1606.07116* (2016).

[28] E. Dennis, A. Kitaev, A. Landahl *et al.* Topological quantum memory. *J. Math. Phys.*, **43**, 4452 (2002).

[29] Y. Aharonov and D. Bohm. Significance of electromagnetic potentials in the quantum theory. *Phys. Rev.*, **115**, 485 (1959).

[30] W. Ehrenberg and R. E. Siday. The refractive index in electron optics and the principles of dynamics. *Proc. Phys. Soc., B*, **62**, 8 (1949).

[31] J. Edmonds. Paths, trees, and flowers. *Can. J. Math.*, **17**, 449 (1965).

[32] V. Kolmogorov. Blossom V: a new implementation of a minimum cost perfect matching algorithm. *Math. Program. Compuation*, **1**, 43 (2009).

[33] S. Bravyi and J. Haah. Quantum self-correction in the 3D cubic code model. *Phys. Rev. Lett.*, **111**, 200501 (2013).

[34] A. G. Fowler. Proof of finite surface code threshold for matching. *Phys. Rev. Lett.*, **109**, 180502 (2012).

[35] A. G. Fowler, A. C. Whiteside, and L. C. L. Hollenberg. Towards practical classical processing for the surface code: Timing analysis. *Phys. Rev. A*, **86**, 042313 (2012).

[36] A. A. Kovalev and L. P. Pryadko. Fault tolerance of quantum low-density parity check codes with sublinear distance scaling. *Phys. Rev. A*, **87**, 020304 (2013).

[37] C. Castelnovo and C. Chamon. Topological order in a three-dimensional toric code at finite temperature. *Phys. Rev. B*, **78**, 155120 (2008).

[38] Z. Nussinov and G. Ortiz. Autocorrelations and thermal fragility of anyonic loops in topologically quantum ordered systems. *Phys. Rev. B*, **77**, 064302 (2008).

[39] A. Kubica, F. Pastawski, and B. Yoshida. Unfolding the color code. *New J. Phys.*, **17**, 083026 (2015).

[40] A. B. Aloshious and P. K. Sarvepalli. Projecting three-dimensional color codes onto three-dimensional toric codes. *Phys. Rev. A*, **98**, 012302 (2018).

[41] N. P. Breuckmann and X. Ni. Scalable Neural Network Decoders for Higher Dimensional Quantum Codes. *Quantum*, **2**, 68 (2018).

[42] K. Duivenvoorden, N. P. Breuckmann, and B. M. Terhal. Renormalization group decoder for a four-dimensional toric code. *IEEE Trans. Inf. Theory*, **65**, 2545 (2019).

[43] A. Kubica and J. Preskill. Cellular-automaton decoders with provable thresholds for topological codes. *Phys. Rev. Lett.*, **123**, 020501 (2019).

[44] A. Kulkarni and P. K. Sarvepalli. Decoding the three-dimensional toric codes and welded codes on cubic lattices. *Phys. Rev. A*, **100**, 012311 (2019).

[45] P. Webster and S. D. Bartlett. Locality-preserving logical operators in topological stabilizer codes. *Phys. Rev. A*, **97**, 012330 (2018).

[46] R. Alicki, M. Horodecki, P. Horodecki *et al.* On thermal stability of topological qubit in Kitaev's 4D model. *Open Syst. Inf. Dyn.*, **17**, 1 (2010).

[47] B. J. Brown, D. Loss, J. K. Pachos *et al.* Quantum memories at finite temperature. *Rev. Mod. Phys.*, **88**, 045005 (2016).

[48] D. Poulin. Stabilizer formalism for operator quantum error correction. *Phys. Rev. Lett.*, **95**, 230504 (2005).

[49] H. Bombín. Single-shot fault-tolerant quantum error correction. *Phys. Rev. X*, **5**, 031043 (2015).

[50] H. Bombín. Resilience to time-correlated noise in quantum computation. *Phys. Rev. X*, **6**, 041034 (2016).

[51] E. Campbell. A theory of single-shot error correction for adversarial noise. *Quantum Sci. Technol.*, **4**, 025006 (2019).

[52] P. W. Shor. Fault-tolerant quantum computation. In *Proc. 37th Annu. Symp. Found. Comput. Sci.*, pages 56–65. IEEE (1996).

[53] D. P. DiVincenzo and P. Aliferis. Effective fault-tolerant quantum computation with slow measurements. *Phys. Rev. Lett.*, **98**, 020501 (2007).

[54] A. M. Steane. Active stabilization, quantum computation, and quantum state synthesis. *Phys. Rev. Lett.*, **78**, 2252 (1997).

[55] A. M. Steane. Fast fault-tolerant filtering of quantum codewords. *arXiv:quant-ph/0202036* (2002).

[56] E. Knill. Scalable quantum computing in the presence of large detected-error rates. *Phys. Rev. A*, **71**, 042322 (2005).

[57] R. Chao and B. W. Reichardt. Quantum error correction with only two extra qubits. *Phys. Rev. Lett.*, **121**, 050502 (2018).

[58] A. Y. Kitaev. Quantum computations: algorithms and error correction. *Russ. Math. Survey*, **52**, 1191 (1997).

[59] A. Y. Kitaev, A. Shen, and M. N. Vyalyi. *Classical and quantum computation.* 47. American Mathematical Society (2002).

[60] G. Nebe, E. M. Rains, and N. J. A. Sloane. The invariants of the Clifford groups. *Des. Codes Cryptogr.*, **24**, 99 (2001).

[61] Y. Shi. Both Toffoli and controlled-NOT need little help to do universal quantum computing. *Quantum Inf. Comput.*, **3**, 84 (2003).

[62] B. Eastin and E. Knill. Restrictions on transversal encoded quantum gate sets. *Phys. Rev. Lett.*, **102**, 110502 (2009).

[63] P. Faist, S. Nezami, V. V. Albert *et al.* Continuous symmetries and approximate quantum error correction. *arXiv:1902.07714* (2019).

[64] R. Raussendorf, J. Harrington, and K. Goyal. A fault-tolerant one-way quantum computer. *Ann. Phys.*, **321**, 2242 (2006).

[65] H. Bombin. Topological order with a twist: Ising anyons from an abelian model. *Phys. Rev. Lett.*, **105**, 030403 (2010).

[66] B. J. Brown, K. Laubscher, M. S. Kesselring *et al.* Poking holes and cutting corners to achieve Clifford gates with the surface code. *Phys. Rev. X*, **7**, 021029 (2017).

[67] C. Horsman, A. G. Fowler, S. Devitt *et al.* Surface code quantum computing by lattice surgery. *New J. Phys.*, **14**, 123011 (2012).

[68] D. Litinski and F. v. Oppen. Lattice surgery with a twist: Simplifying Clifford gates of surface codes. *Quantum*, **2**, 62 (2018).

[69] D. Litinski. A game of surface codes: Large-scale quantum computing with lattice surgery. *Quantum*, **3**, 128 (2019).

[70] A. G. Fowler and C. Gidney. Low overhead quantum computation using lattice surgery. *arXiv:1808.06709* (2018).

[71] S. Bravyi and J. Haah. Magic-state distillation with low overhead. *Phys. Rev. A*, **86**, 052329 (2012).

[72] A. Paetznick and B. W. Reichardt. Universal fault-tolerant quantum computation with only transversal gates and error correction. *Phys. Rev. Lett.*, **111**, 090505 (2013).

[73] S. Bravyi and A. Kitaev. Universal quantum computation with ideal Clifford gates and noisy ancillas. *Phys. Rev. A*, **71**, 022316 (2005).

[74] E. T. Campbell and M. Howard. Unified framework for magic state distillation and multiqubit gate synthesis with reduced resource cost. *Phys. Rev. A*, **95**, 022316 (2017).

[75] E. T. Campbell and M. Howard. Unifying gate synthesis and magic state distillation. *Phys. Rev. Lett.*, **118**, 060501 (2017).

[76] J. Haah and M. B. Hastings. Codes and protocols for distilling T, controlled-S, and Toffoli gates. *Quantum*, **2**, 71 (2018).

[77] D. Gottesman and I. L. Chuang. Demonstrating the viability of universal quantum computation using teleportation and single-qubit operations. *Nature*, **402**, 390 (1999).

[78] X. Zhou, D. W. Leung, and I. L. Chuang. Methodology for quantum logic gate construction. *Phys. Rev. A*, **62**, 052316 (2000).

[79] D. Aharonov and M. Ben-Or. Fault-tolerant quantum computation with constant error rate. In *Proc. 29th Annual IEEE Symp. on the Theory of Computer Science (STOC'97)*, pages 176–188. ACM (1997).

[80] A. Y. Kitaev. Quantum computations: algorithms and error correction. *Russian Math. Surveys*, **52**, 1191 (1997).

[81] E. Knill, R. Laflamme, and W. H. Zurek. Resilient quantum computation: error models and thresholds. *Proc. Royal Soc. Lond., Ser. A: Math. Phys. Eng. Sci.*, **454**, 365 (1998).

[82] D. Aharonov, A. Kitaev, and J. Preskill. Fault-tolerant quantum computation with long-range correlated noise. *Phys. Rev. Lett.*, **96**, 050504 (2006).

[83] P. Aliferis and J. Preskill. Fault-tolerant quantum computation against biased noise. *Phys. Rev. A*, **78**, 052331 (2008).

[84] H. K. Ng and J. Preskill. Fault-tolerant quantum computation versus gaussian noise. *Phys. Rev. A*, **79**, 032318 (2009).

[85] J. Preskill. Sufficient condition on noise correlations for scalable quantum computing. *Quantum Inf. Comput.*, **13**, 181 (2013).

[86] J. O'Gorman and E. T. Campbell. Quantum computation with realistic magic-state factories. *Phys. Rev. A*, **95**, 032338 (2017).

[87] M. B. Hastings and J. Haah. Distillation with sublogarithmic overhead. *Phys. Rev. Lett.*, **120**, 050504 (2018).

[88] S. Bravyi and R. König. Classification of topologically protected gates for local stabilizer codes. *Phys. Rev. Lett.*, **110**, 170503 (2013).

[89] F. Pastawski and B. Yoshida. Fault-tolerant logical gates in quantum error-correcting codes. *Phys. Rev. A*, **91**, 012305 (2015).

[90] T. Jochym-O'Connor, A. Kubica, and T. J. Yoder. Disjointness of stabilizer codes and limitations on fault-tolerant logical gates. *Phys. Rev. X*, **8**, 021047 (2018).

[91] B. Yoshida. Topological color code and symmetry-protected topological phases. *Phys. Rev. B*, **91**, 245131 (2015).

[92] M. Beverland, O. Buerschaper, R. König *et al.* Protected gates for topological quantum field theories. *J. Math. Phys.*, **57** (2016).

[93] H. Bombín and M. A. Martin-Delgado. Topological quantum distillation. *Phys. Rev. Lett.*, **97**, 180501 (2006).

[94] H. Bombín and M. A. Martin-Delgado. Topological computation without braiding. *Phys. Rev. Lett.*, **98**, 160502 (2007).

[95] H. Bombin and M. A. Martin-Delgado. Exact topological quantum order in $D = 3$ and beyond: Branyons and brane-net condensates. *Phys. Rev. B*, **75**, 075103 (2007).

[96] A. J. Landahl, J. T. Anderson, and P. R. Rice. Fault-tolerant quantum computing with color codes. *arXiv:1108.5738* (2011).

[97] H. Bombín. Gauge color codes: optimal transversal gates and gauge fixing in topological stabilizer codes. *New J. Phys.*, **17**, 083002 (2015).

[98] A. Kubica and M. E. Beverland. Universal transversal gates with color codes: A simplified approach. *Phys. Rev. A*, **91**, 032330 (2015).

[99] F. H. E. Watson, E. T. Campbell, H. Anwar *et al.* Qudit color codes and gauge color codes in all spatial dimensions. *Phys. Rev. A*, **92**, 022312 (2015).

[100] D. S. Wang, A. G. Fowler, C. D. Hill *et al.* Graphical algorithms and threshold error rates for the 2D colour code. *Quantumt Inf. Comput.*, **10**, 780 (2010).

[101] H. Bombín. Dimensional jump in quantum error correction. *New J. Phys.*, **18**, 043038 (2016).

[102] B. J. Brown, N. H. Nickerson, and D. E. Browne. Fault-tolerant error correction with the gauge color code. *Nat. Commun.*, **7**, 12302 (2016).

[103] R. Barends, J. Kelly, A. Megrant *et al.* Superconducting quantum circuits at the surface code threshold for fault tolerance. *Nature*, **508**, 500 (2014).

[104] K. Wright, K. Beck, S. Debnath *et al.* Benchmarking an 11-qubit quantum computer. *arXiv:1903.08181* (2019).

[105] H. Bombín, G. Duclos-Cianci, and D. Poulin. Universal topological phase of two-dimensional stabilizer codes. *New J. Phys.*, **14**, 073048 (2012).

[106] B. Yoshida. Classification of quantum phases and topology of logical operators in an exactly solved model of quantum codes. *Ann. Phys.*, **326**, 15 (2011). January 2011 Special Issue.

[107] H. Bombín. Structure of 2D topological stabilizer codes. *Commun. Math. Phys.*, **327**, 387 (2014).

[108] A. B. Aloshious, A. N. Bhagoji, and P. K. Sarvepalli. On the local equivalence of 2D color codes and surface codes with applications. *arXiv:1804.00866* (2018).

[109] N. Delfosse. Decoding color codes by projection onto surface codes. *Phys. Rev. A*, **89**, 012317 (2014).

[110] A. Kubica and N. Delfosse. Efficient color code decoders in $d \geq 2$ dimensions from toric code decoders. *arXiv:1905.07393* (2019).

[111] J. E. Moussa. Transversal clifford gates on folded surface codes. *Phys. Rev. A*, **94**, 042316 (2016).

[112] M. Vasmer and D. E. Browne. Three-dimensional surface codes: Transversal gates and fault-tolerant architectures. *Phys. Rev. A*, **100**, 012312 (2019).

[113] H. S. M. Coxeter. *Regular polytopes.* Dover Publications, Inc., 3 edition (1973).

[114] X.-G. Wen. Quantum orders in an exact soluble model. *Phys. Rev. Lett.*, **90**, 016803 (2003).

[115] J. T. Anderson. Homological stabilizer codes. *Ann. Phys.*, **330**, 1 (2013).

[116] Y. Tomita and K. M. Svore. Low-distance surface codes under realistic quantum noise. *Phys. Rev. A*, **90**, 062320 (2014).

[117] T. J. Yoder and I. H. Kim. The surface code with a twist. *Quantum*, **1**, 2 (2017).

[118] M. Li, D. Miller, M. Newman *et al.* 2d compass codes. *Phys. Rev. X*, **9**, 021041 (2019).

[119] R. Webb. Stella: polyhedron navigator. *Symmetry: Culture and Science*, **11**, 231 (2000).

[120] J. H. Conway, H. Burgiel, and C. Goodman-Strauss. *The symmetries of things.* AK Peters/CRC Press (2016).

[121] A. Kubica (2018). Private communication.

[122] D. Bacon. Operator quantum error-correcting subsystems for self-correcting quantum memories. *Phys. Rev. A*, **73**, 012340 (2006).

[123] B. Criger and B. Terhal. Noise thresholds for the [[4, 2, 2]]-concatenated toric code. *Quantum Inf. Comput.*, **16**, 1261 (2016).

[124] E. Cambpell. The smallest interesting colour code. `https://earltcampbell.com/2016/09/26/the-smallest-interesting-colour-code/` (2016). Accessed: 17-07-2019.

[125] W. Wijthoff. A relation between the polytopes of the C600-family. *Koninklijke Nederlandse Akademie van Wetenschappen, Proc. Ser. B Phys. Sci.*, **20**, 966 (1918).

[126] C. Vuillot and N. P. Breuckmann. Quantum pin codes. *arXiv:1906.11394* (2019).

[127] M. H. Freedman, D. A. Meyer, and F. Luo. Z2-systolic freedom and quantum codes. *Mathematics of quantum computation, Chapman & Hall/CRC*, pages 287–320 (2002).

[128] L. Guth and A. Lubotzky. Quantum error correcting codes and 4-dimensional arithmetic hyperbolic manifolds. *J. Math. Phys.*, **55**, 082202 (2014).

[129] N. P. Breuckmann and B. M. Terhal. Constructions and noise threshold of hyperbolic surface codes. *IEEE Trans. Inf. Theory*, **62**, 3731 (2016).

[130] N. P. Breuckmann, C. Vuillot, E. Campbell *et al.* Hyperbolic and semi-hyperbolic surface codes for quantum storage. *Quantum Sci. Technol.*, **2**, 035007 (2017).

[131] V. Londe and A. Leverrier. Golden codes: quantum LDPC codes built from regular tessellations of hyperbolic 4-manifolds. *Quantum Inf. Comput.*, **19**, 0361 (2019).

[132] J. E. Humphreys. *Reflection groups and Coxeter groups.* Cambridge University Press (1992).

[133] H. G. Katzgraber, H. Bombin, and M. A. Martin-Delgado. Error threshold for color codes and random three-body Ising models. *Phys. Rev. Lett.*, **103**, 090501 (2009).

[134] A. Kubica, M. E. Beverland, F. Brandão *et al.* Three-dimensional color code thresholds via statistical-mechanical mapping. *Phys. Rev. Lett.*, **120**, 180501 (2018).

[135] C. T. Chubb and S. T. Flammia. Statistical mechanical models for quantum codes with correlated noise. *arXiv:1809.10704* (2018).

[136] T. Ohno, G. Arakawa, I. Ichinose *et al.* Phase structure of the random-plaquette Z2 gauge model: accuracy threshold for a toric quantum memory. *Nucl. Phys. B*, **697**, 462 (2004).

[137] Y. Ozeki and N. Ito. Multicritical dynamics for the $\pm J$ Ising model. *J. Phys. A: Math. Gen.*, **31**, 5451 (1998).

[138] M. Hasenbusch, F. P. Toldin, A. Pelissetto *et al.* Magnetic-glassy multicritical behavior of the three-dimensional $\pm J$ Ising model. *Phys. Rev. B*, **76**, 184202 (2007).

[139] D. S. Wang, A. G. Fowler, A. M. Stephens *et al.* Threshold error rates for the toric and surface codes. *Quantum Inf. Comput.*, **10**, 456 (2010).

[140] A. A. Hagberg, D. A. Schult, and P. J. Swart. Exploring network structure, dynamics, and function using networkx. In *Proc. 7th Python Sci. Conf. (SciPy2008)*, pages 11–15 (2008).

[141] C. Wang, J. Harrington, and J. Preskill. Confinement-Higgs transition in a disordered gauge theory and the accuracy threshold for quantum memory. *Ann. Phys.*, **303**, 31 (2003).

[142] N. Delfosse and J. Tillich. A decoding algorithm for CSS codes using the X/Z correlations. In *2014 IEEE Int. Symp. Inf. Theory*, pages 1071–1075 (2014).

[143] G. Duclos-Cianci and D. Poulin. Fast decoders for topological quantum codes. *Phys. Rev. Lett.*, **104**, 050504 (2010).

[144] M. Herold, E. T. Campbell, J. Eisert *et al.* Cellular-automaton decoders for topological quantum memories. *Npj Quantum Inf.*, **1**, 15010 (2015). Article.

[145] M. Herold, M. J. Kastoryano, E. T. Campbell *et al.* Cellular automaton decoders of topological quantum memories in the fault tolerant setting. *New J. Phys.*, **19**, 063012 (2017).

[146] N. Delfosse and N. H. Nickerson. Almost-linear time decoding algorithm for topological codes. *arXiv:1709.06218* (2017).

[147] A. L. Toom. Stable and attractive trajectories in multicomponent systems. *Adv. Probab.*, **6**, 549 (1980).

[148] C. H. Bennett and G. Grinstein. Role of irreversibility in stabilizing complex and nonergodic behavior in locally interacting discrete systems. *Phys. Rev. Lett.*, **55**, 657 (1985).

[149] G. Grinstein. Can complex structures be generically stable in a noisy world? *IBM J. Res. Dev.*, **48**, 5 (2004).

[150] J. W. Harrington. *Analysis of quantum error-correcting codes: symplectic lattice codes and toric codes*. Ph.D. thesis, Caltech (2004).

[151] N. P. Breuckmann, K. Duivenvoorden, D. Michels *et al.* Local decoders for the 2D and 4D toric code. *Quantum Inf. Comput.*, **17**, 0181 (2017).

[152] G. Dauphinais and D. Poulin. Fault-tolerant quantum error correction for non-abelian anyons. *Commun. Math. Phys.*, **355**, 519 (2017).

[153] F. Pastawski, L. Clemente, and J. I. Cirac. Quantum memories based on engineered dissipation. *Phys. Rev. A*, **83**, 012304 (2011).

[154] A. 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*. Ph.D. thesis, Caltech (2018).

[155] P. Gács and J. Reif. A simple three-dimensional real-time reliable cellular array. *J. Comput. Syst. Sci.*, **36**, 125 (1988).

[156] J. Van Den Berg and H. Kesten. Inequalities with applications to percolation and reliability. *Journal of Applied Probability*, **22**, 556 (1985).

[157] O. Fawzi, A. Grospellier, and A. Leverrier. Constant overhead quantum fault-tolerance with quantum expander codes. In *2018 IEEE 59th Annu. Symp. Found. Comput. Sci. (FOCS)*, pages 743–754 (2018).

[158] P. Aliferis, F. Brito, D. P. DiVincenzo *et al.* Fault-tolerant computing with biased-noise superconducting qubits: a case study. *New J. Phys.*, **11**, 013061 (2009).

[159] M. D. Shulman, O. E. Dial, S. P. Harvey *et al.* Demonstration of entanglement of electrostatically coupled singlet-triplet qubits. *Science*, **336**, 202 (2012).

[160] D. Nigg, M. Mueller, E. A. Martinez *et al.* Quantum computations on a topologically encoded qubit. *Science*, **345**, 302 (2014).

[161] S. Crain, C. Cahall, G. Vrijsen *et al.* High-speed, low-crosstalk detection of a trapped $^{171}\mathrm{Yb}^{+}$ ion ancilla qubit using superconducting nanowire single photon detectors. *arXiv:1902.04059* (2019).

[162] N. P. Breuckmann. *Homological quantum codes beyond the toric code*. Ph.D. thesis, RWTH Aachen (2018).

[163] D. Herr, F. Nori, and S. J. Devitt. Lattice surgery translation for quantum computation. *New J. Phys.*, **19**, 013034 (2017).

[164] S. D. Barrett and P. Kok. Efficient high-fidelity quantum computation using matter qubits and linear optics. *Phys. Rev. A*, **71**, 060310 (2005).

[165] K. Fujii, T. Yamamoto, M. Koashi *et al.* A distributed architecture for scalable quantum computation with realistically noisy devices. *arXiv:1202.6588* (2012).

[166] N. H. Nickerson, Y. Li, and S. C. Benjamin. Topological quantum computing with a very noisy network and local error rates approaching one percent. *Nat. Commun.*, **4**, 1756 (2013).

[167] C. Monroe, R. Raussendorf, A. Ruthven *et al.* Large-scale modular quantum-computer architecture with atomic memory and photonic interconnects. *Phys. Rev. A*, **89**, 022317 (2014).

[168] N. H. Nickerson, J. F. Fitzsimons, and S. C. Benjamin. Freely scalable quantum technologies using cells of 5-to-50 qubits with very lossy and noisy photonic links. *Phys. Rev. X*, **4**, 041041 (2014).

[169] N. Kalb, A. A. Reiserer, P. C. Humphreys *et al.* Entanglement distillation between solid-state quantum network nodes. *Science*, **356**, 928 (2017).

[170] K. Kieling, T. Rudolph, and J. Eisert. Percolation, renormalization, and quantum computing with nondeterministic gates. *Phys. Rev. Lett.*, **99**, 130501 (2007).

[171] M. Gimeno-Segovia, P. Shadbolt, D. E. Browne *et al.* From three-photon Greenberger-Horne-Zeilinger states to ballistic universal quantum computation. *Phys. Rev. Lett.*, **115**, 020502 (2015).

[172] C. Sun, M. T. Wade, Y. Lee *et al.* Single-chip microprocessor that communicates directly using light. *Nature*, **528**, 534 (2015).

[173] T. Rudolph. Why I am optimistic about the silicon-photonic route to quantum computing. *APL Photonics*, **2**, 030901 (2017).

[174] B. J. Brown. A fault-tolerant non-Clifford gate for the surface code in two dimensions. *arXiv:1903.11634* (2019).

[175] H. Bombin. 2D quantum computation with 3D topological codes. *arXiv:1810.09571* (2018).

[176] S. Bravyi, G. Duclos-Cianci, D. Poulin *et al.* Subsystem surface codes with three-qubit check operators. *Quantum Inf. Comput.*, **13**, 963 (2013).

[177] S. Burton. Spectra of gauge code hamiltonians. *arXiv:1801.03243* (2018).

[178] S. Roberts and S. D. Bartlett. Symmetry-protected self-correcting quantum memories. *arXiv:1805.01474* (2018).