Three-dimensional surface codes: Transversal gates and fault-tolerant architectures

Michael Vasmer* and Dan E. Browne

Department of Physics and Astronomy, University College London, Gower Street, London, WC1E 6BT, United Kingdom

(Received 23 April 2018; published 9 July 2019)

One of the leading quantum computing architectures is based on the two-dimensional (2D) surface code. This code has many advantageous properties such as a high error threshold and a planar layout of physical qubits where each physical qubit need only interact with its nearest neighbors. However, the transversal logical gates available in 2D surface codes are limited. This means that an additional (resource-intensive) procedure known as magic state distillation is required to do universal quantum computing with 2D surface codes. Here, we examine three-dimensional (3D) surface codes in the context of quantum computation. We introduce a picture for visualizing 3D surface codes which is useful for analyzing stacks of three 3D surface codes. We use this picture to prove that the CZ and CCZ gates are transversal in 3D surface codes. We also generalize the techniques of 2D surface-code lattice surgery to 3D surface codes. We combine these results and propose two quantum computing architectures based on 3D surface codes. Magic state distillation is not required in either of our architectures. Finally, we show that a stack of three 3D surface codes can be transformed into a single 3D color code (another type of quantum error-correcting code) using code concatenation.

DOI: 10.1103/PhysRevA.100.012312

I. INTRODUCTION

The family of quantum error-correcting codes known as surface codes (also called toric codes or homological codes) have generated a great deal of theoretical and experimental interest since their introduction by Kitaev [1]. We can define surface codes in any spatial dimension $D \geqslant 2$. The two-dimensional (2D) surface code [2,3] is the basis of one of the leading proposals for a fault-tolerant quantum computing architecture [4]. The biggest advantage of the 2D surface code is its high-error threshold which approaches 1% [5–7], a value which has been achieved in various qubit technologies [8,9]. The other main advantage of the 2D surface code is that it has a simple structure consisting of a planar layout of qubits where each qubit only needs to interact with four neighboring qubits. Experimental groups in universities and industry are targeting the surface code as their eventual fault-tolerant architecture [8,10–13]. However, these groups are still a long way off the millions of qubits required to run quantum algorithms such as Shor's algorithm [14] on a surface-code quantum computer [4,15]. One of the contributing factors to the large qubit overhead of 2D surface-code architectures is that a procedure known as magic state distillation is needed if we want to implement the non-Clifford T gate [16] in 2D surface codes. Non-Clifford gates are required for universal quantum computation but they are rarely easy to implement in quantum error-correcting codes. Magic state distillation is estimated to have a resource cost ∼150–300 times greater than the resource cost of realizing the control-NOT (CNOT) gate in 2D surface-code architectures [15]. The overhead associated with magic state distillation has motivated research into alternative methods for realizing non-Clifford gates in topological codes. For example, the 3D gauge color code has a transversal non-Clifford gate and this code forms the basis of a universal quantum computing architecture with attractive properties such as single-shot error correction [17–19]. The resource overheads of 3D gauge color codes and 2D surface codes with magic state distillation are estimated to scale in a similar way [15], so for different ranges of parameters either option could be advantageous. However, it has been argued that 2D surface-code architectures will be superior for current experimental parameters due to the superlative error threshold of 2D surface codes [15].

In this article, we study three-dimensional (3D) surface codes. These codes were first introduced in [3] and their topological entropy was studied in [20]. Most previous work on 3D surface codes in the context of quantum computing has concentrated on the relationship between 3D surface codes and 3D color codes. Color codes are another family of topological error-correcting codes which share some features with surface codes. It turns out that we can transform any 3D color code into three 3D surface codes using local Clifford unitaries [21]. This relationship has implications for quantum computing with 3D surface codes and 3D color codes. For example, using the mapping between the two code families, we can use 3D surface-code decoders to decode 3D color codes [22]. This is useful because efficient 3D color-code decoders are difficult to construct. Color codes tend to have a larger range of transversal logical gates when compared with surface codes [23,24]. This implies that we can use the relationship between surface codes and color codes to realize logical gates in 3D surface codes which are not naively available. Indeed, we can use the mapping between the two code families to implement a locality-preserving control-control-Z (CCZ) gate in the 3D surface code [21]. Locality-preserving logical operators (LPLOs) are naturally fault-tolerant because the growth of errors under a LPLO is bounded by a constant [25,26].

*michael.vasmer.15@ucl.ac.uk

Recently, the LPLOs of 3D surface codes with different boundary conditions were classified using a correspondence between logical operators and domain walls [26].

Here, we introduce a way of visualizing 3D surface codes, which we call the rectified picture. We use the rectified picture to analyze stacks of three 3D surface codes. We show that 3D surface codes possess more transversal gates than was previously thought, namely, that both the control-Z (CZ) and CCZ gates are transversal for stacks of three 3D surface codes. These results build on the results in [21] and [26], where it was shown that CZ and CCZ are LPLOs for stacks of three 3D surface codes. We also show that the mapping between 3D surface codes and 3D color codes described in [21] can be achieved using code concatenation. This result generalizes the code concatenation transformations for 2D surface codes and 2D color codes presented in [27]. The second focus of this article is on quantum computing architectures based on 3D surface codes. We propose a hybrid 2D-3D surface-code architecture and a purely 3D surface-code architecture. Both of these architectures use the techniques of lattice surgery [28], which we generalize to 3D surface codes. Our architectures achieve universal quantum computation without needing magic state distillation. It is possible that these architectures may require fewer resources than 2D surface-code architectures in certain systems. For example, one could imagine taking advantage of the connections between qubits allowed in a modular architecture [29–33] to build a code which is local in three spatial dimensions. However, more research is necessary before we can definitively assess the resource costs of our proposed architectures.

The remainder of this article is structured as follows. We provide background information on topological codes in Sec. II and we introduce our rectified picture of 3D surface codes in Sec. III. In Sec. IV, we detail a concatenation transformation that maps three 3D surface codes to a 3D color code. In Sec. V, we show that CZ and CCZ are transversal for stacks of three 3D surface codes and we explain how to implement a universal gate set. In Secs. VI and VII, we discuss 3D surface-code lattice surgery and universal quantum computing architectures which utilize 3D surface codes. Finally, in Sec. VIII, we discuss the implications of our work and outline future research directions.

II. BACKGROUND

Surface codes are a family of topological stabilizer codes [1–3]. A stabilizer code is a quantum error-correcting code defined by its stabilizer group $\mathcal{S}$, an Abelian subgroup of the Pauli group where $-I \notin \mathcal{S}$ [34]. Every encoded state $|\overline{\psi}\rangle$ in the code is stabilized by $\mathcal{S}$, that is, $\forall\, S \in \mathcal{S}$, $S\,|\overline{\psi}\rangle = |\overline{\psi}\rangle$. We summarize the properties of a quantum error-correcting code with the shorthand notation $[[n, k, d]]$, where n is the number of physical qubits, k is the number of encoded logical qubits, and d is the code distance. The code distance of a quantum error-correcting code is equal to the weight of the minimum weight logical operator of the code. The weight of an operator is simply the number of qubits it acts on nontrivially.

A topological code is a code defined on some lattice with physical qubits placed on some of the elements of the lattice (the edges, for example). The stabilizers of a topological code act in geometrically local regions and the logical operators of the code form topologically nontrivial paths or surfaces on the lattice. We are particularly interested in two types of logical operators in topological codes: locality-preserving logical operators (LPLOs) and transversal logical operators. LPLOs are operators that map errors in some region of a code R to errors in a region R' which is at most a constant size C bigger than R [26]. A transversal logical operator is a logical operator realized by a quantum circuit of depth one which does not couple physical qubits in the same code (block). Transversal logical operators are LPLOs because transversal logical operators never spread errors from one physical qubit to another physical qubit in the same code.

In this article, we consider 2D and 3D surface codes. We begin by defining 2D surface codes in what we call the "Kitaev picture." This is the formalism introduced by Kitaev in [1]. We place qubits on the edges of a 2D lattice. We associate Z stabilizers with the faces of the lattice and X stabilizers with the vertices of the lattice. That is, for each face f we have a stabilizer $S_f = \bigotimes_{e\in f} Z(q_e)$ where $Z(q_e)$ denotes a Z operator applied to the qubit on edge e. Analogously, for each vertex v we have a stabilizer $S_v = \bigotimes_{e:v\in e} X(q_e)$. We interpret unsatisfied stabilizers (stabilizers with -1 eigenvalues) as quasiparticles. Following the convention in the literature [3], we refer to unsatisfied X stabilizers as electric charges (e) and unsatisfied Z stabilizers as magnetic fluxes (m). In the 2D surface code, both e and m quasiparticles are zero-dimensional (0D) objects. In the bulk of the lattice, we can only create or destroy pairs of quasiparticles of the same type. 2D surface codes can have two types of boundary: rough boundaries and smooth boundaries. Single e quasiparticles can condense on the rough boundaries and single m quasiparticles can condense on the smooth boundaries. In this context, quasiparticle condensation means that a single quasiparticle can be created or destroyed at the relevant boundary. In the 2D surface code, logical $\overline{Z}$ operators are strings of Z operators from one rough boundary to another and logical $\overline{X}$ operators are strings of X operators from one smooth boundary to another.

There is an equivalent picture of 2D surface codes which is related to the Kitaev picture by a medial transformation [35–37]. This picture is often called the rotated picture [38]. In the rotated picture, qubits are on vertices and stabilizers are associated with faces. Rotated picture lattices are 2-face-colorable, i.e., every face in the lattice can be assigned one of two colors such that no faces which share an edge have the same color. In this picture, we associate Z stabilizers with c colored faces (c faces) and X stabilizers with c' faces. For example, the stabilizer associated with the c face f_c is $S_{f_c} = \bigotimes_{v\in f_c} Z(v)$. Figure 1 shows two distance three 2D surface codes in the Kitaev picture and the rotated picture.

We now turn to 3D surface codes. Initially, we define 3D surface codes in the Kitaev picture [3], using the same conventions as the 2D surface code. We place qubits on the edges of a 3D lattice, we associate X stabilizers with the vertices of the lattice and we associate Z stabilizers with the faces of the lattice. We again interpret unsatisfied X (Z) stabilizers as e (m) quasiparticles. However, in contrast to 2D surface codes, in the 3D surface code m quasiparticles are 1D objects (e quasiparticles are still 0D). 3D surface

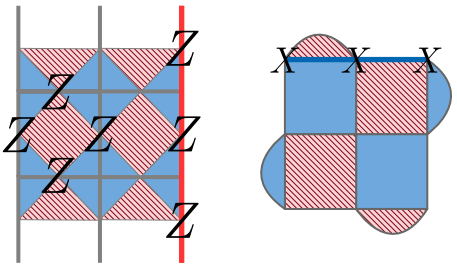


FIG. 1. 2D surface codes in the Kitaev picture and the rotated picture. On the left we show the [[13,1,3]] surface code in the Kitaev picture and the rotated picture [red (hatched) and blue (solid) lattice]. We highlight a Z stabilizer and $\overline{Z}$ operator. On the right we show the [[9,1,3]] surface code in the rotated picture. We highlight an $\overline{X}$ operator. In both codes, the top and bottom boundaries are rough boundaries and the left and right boundaries are smooth boundaries.

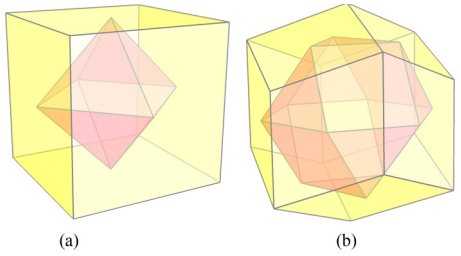


FIG. 2. Polyhedra and their duals. (a) A cube [yellow (light gray)] and its dual octahedron [red (medium gray)]. (b) A rhombic dodecahedron [yellow (light gray)] and its dual cuboctahedron [red (medium gray)].

codes also have rough and smooth boundaries which are again defined by quasiparticle condensation. As in the 2D case, e (m) quasiparticles can condense on rough (smooth) boundaries. $\overline{Z}$ operators in 3D surface codes are strings of Z operators which terminate at different rough boundaries. $\overline{X}$ operators are membranes of X operators with a boundary which spans contiguous smooth boundaries. In this article we only consider 3D surface codes with six boundaries (two rough boundaries and four smooth boundaries) where the rough boundaries are on opposite sides of the lattice.

So far, we have only discussed the primal lattice picture of 3D surface codes. We can also analyze 3D surface codes in the dual lattice picture. Given a 3D lattice, we can construct its dual using a simple procedure. First, we create vertices at the center of the cells of the original lattice. Next, we join these vertices with edges if their corresponding cells in the original lattice shared a face. Finally, we delete the original (primal) lattice. This transformation maps vertices to cells, edges to faces, faces to edges, and cells to vertices. Therefore, in the dual lattice picture of 3D surface codes, qubits are placed on the faces, X stabilizers are associated with cells, and Z stabilizers are associated with edges. In the remainder of this article, we will use both the primal lattice picture and the dual lattice picture to analyze 3D surface codes.

III. RECTIFIED PICTURE

In this section, we describe a picture which we use to analyze stacks of 3D surface codes. We call this picture the "rectified picture." This picture is a generalization of the rotated picture of 2D surface codes and it is similar to the primal lattice picture of 3D color codes [23]. We start with a 3D surface-code primal lattice in the Kitaev picture. To transform to the rectified picture, we rectify the primal lattice. A rectification (or full truncation) is a geometric transformation where the edges of a lattice are truncated to points [39]. Specifically, to perform a rectification, we use the following procedure. First, we create vertices at the midpoints of the edges of the original lattice. Next, we join these vertices with edges if their corresponding edges in the original lattice were part of the same face. Finally, we delete the original lattice to obtain the rectified lattice. Under a rectification, 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 stabilizers are associated with cells, and Z stabilizers are associated with faces. We note that there is an analogous transformation which maps a 3D surface-code Kitaev picture dual lattice to a rectified picture lattice. This transformation is called a face rectification and is equivalent to taking the dual of every cell in the lattice. Given a polyhedral cell, we construct its dual by creating vertices at the center of the original polyhedron's faces. We then connect these vertices with edges if their corresponding faces in the original polyhedron share an edge. Figure 2 shows a cube and a cuboctahedron along with their dual polyhedra.

The utility of the rectified picture comes when we consider stacks of three 3D surface codes. This is because different lattices in the Kitaev picture correspond to the same lattice in the rectified picture. Hence, instead of analyzing three overlapping surface-code lattices in the Kitaev picture, we can analyze a single lattice in the rectified picture. In this article, we concentrate on surface codes defined on cubic lattices and tetrahedral-octahedral lattices (primal lattices in the Kitaev picture). In the familiar cubic lattice, eight cubes meet at every vertex. In the tetrahedral-octahedral lattice, six octahedra and eight tetrahedra meet at every vertex. The cubic lattice is self-dual and the dual of a tetrahedral-octahedral lattice is a rhombic dodecahedral lattice (a lattice where every cell is a rhombic dodecahedron). Let us consider how the cubic lattice is transformed under rectification: vertices are mapped to octahedra and cubes are mapped to cuboctahedra. Cuboctahedra are polyhedra with 12 vertices where two triangle faces and two square faces meet at each vertex. Figure 2(b) shows a cuboctahedron. The rectification of a cubic lattice is usually called the rectified cubic lattice. In a rectified cubic lattice, two octahedra and four cuboctahedra meet at every vertex. Figure 3 shows a portion of a rectified cubic lattice.

Next, we consider the rectification of a tetrahedral-octahedral lattice. Under rectification, octahedra transform into cuboctahedra, tetrahedra transform into octahedra, and vertices become cuboctahedra. Therefore, rectification also transforms the tetrahedral-octahedral lattice into a rectified cubic lattice. In fact, we can arrange one cubic lattice and two tetrahedral-octahedral lattices in such a way that all three lattices are transformed into the exact same rectified

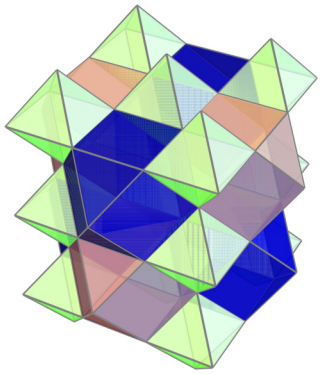
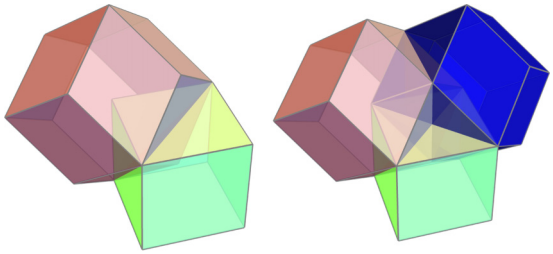


FIG. 3. Part of a rectified cubic lattice. Rectified cubic lattices consist of cuboctahedra [blue (dark gray) and red (medium gray)] and octahedra [green (light gray)]. Four cuboctahedra and two octahedra meet at each vertex.



(a) (b)

FIG. 4. Arranging a cubic lattice and two rhombic dodecahedral lattices such that they are transformed to the same rectified cubic lattice under face rectification. We show a single cell from each lattice. (a) We arrange a rhombic dodecahedral lattice [red (medium gray)] and a cubic lattice [green (light gray)] such that the acute vertices of the rhombic dodecahedra occupy the same locations as the a vertices of the cubes. In this arrangement, the b vertices of the cubes lie at the center of the rhombic dodecahedra and the obtuse vertices of the rhombic dodecahedra lie at the center of the cubes. (b) We add a second rhombic dodecahedral lattice [blue (dark gray)] and arrange it such that the acute vertices of the rhombic dodecahedra occupy the same locations as the b vertices of the cubes. In this arrangement, the obtuse vertices of the rhombic dodecahedra lie at the center of the cubes and the a vertices of the cubes lie at the center of the rhombic dodecahedra.

cubic lattice under rectification. To see how this works, it is easiest to consider the dual lattices and the face-rectification transformation. As we mentioned earlier, the dual of a tetrahedral-octahedral lattice is a rhombic dodecahedral lattice. A rhombic dodecahedron is a polyhedron with 12 rhombic faces. Rhombic dodecahedra have two different types of vertex. Acute vertices are the points where the acute angle corners of four rhombi meet whereas obtuse vertices are the points where the obtuse angle corners of three rhombi meet. Figure 2(b) shows a rhombic dodecahedron. In a rhombic dodecahedral lattice, four rhombic dodecahedra meet at every obtuse vertex and six rhombic dodecahedra meet at every acute vertex.

We now show how to arrange one cubic lattice and two rhombic dodecahedral lattices such that they are mapped to exactly the same lattice under face rectification. This fact is the reason we can define three surface codes on the same rectified cubic lattice. We assume the three lattices are infinite for simplicity. First, we note that the cubic lattice is 2-vertex-colorable, i.e., all the vertices in the cubic lattice can be assigned a color such that no vertices which share an edge have the same color. We give these two sets of vertices the labels a and b. We arrange the cubic lattice and one of the rhombic dodecahedral lattices such that the acute vertices of the rhombic dodecahedra occupy the same positions as the a vertices of the cubes. In this arrangement, the obtuse vertices of the rhombic dodecahedra are at the center of cubes and the b vertices of the cubes are at the center of rhombic dodecahedra. This layout is shown (for a single cube and rhombic dodecahedron) in Fig. 4(a). Next, we add a second rhombic dodecahedral lattice and arrange it such that its acute vertices occupy the same positions as the b vertices of the cubes. The arrangement of all three lattices is illustrated in Fig. 4(b).

In the arrangement of lattices we have just described, all three lattices will be mapped to an identical rectified cubic lattice by the face-rectification transformation. To see why this is true we consider how the cells and vertices of the lattices transform. The cubes and the obtuse vertices of the rhombic dodecahedra both transform into octahedra. The obtuse vertices of the rhombic dodecahedra lie at the center of the cubes in our arrangement so each lattice transforms in the same way at these positions. Similarly, the acute vertices of one rhombic dodecahedral lattice occupy the same position as the a vertices of the cubic lattice. Both these types of vertices lie at the center of the cells of the other rhombic dodecahedral lattice. Rhombic dodecahedra, acute vertices, and vertices of cubes are all mapped to cuboctahedra under face rectification. So the three lattices transform in the same way at these positions. An identical argument holds for the b vertices of the cubes.

A. A family of stacked 3D surface codes

In this section, we define a family of stacked 3D surface codes. We call these codes rectified cubic codes. Each member of the family consists of three 3D surface codes supported on the same rectified cubic lattice. We first discuss the structure of the rectified cubic lattices, then we define the surface codes.

1. Lattice structure

Rectified cubic lattices are 3-cell-colorable, and we color the cells of our lattices with the colors $\{r, g, b\}$. We assume that octahedra are colored g and the two sets of cuboctahedra are colored r and b. We assign each lattice face the color of the two cells it is part of. For example, a face shared by a r cell and a g cell is a rg face. A face on a boundary which is only part of one cell is assigned the combination of colors it would have in a 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

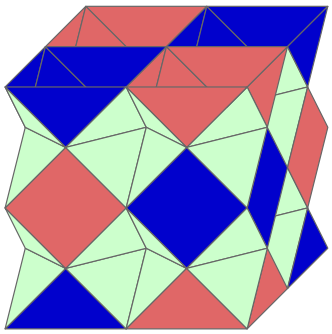


FIG. 5. The $d = 3$ rectified cubic lattice. The top and bottom boundaries are half-cuboctahedra boundaries whereas the other four boundaries are full-cuboctahedra boundaries.

boundaries half cuboctahedra boundaries and full cuboctahedra boundaries, respectively. Each lattice in the family has two half-cuboctahedra boundaries and four full-cuboctahedra boundaries. Opposite boundaries are the same type. The two types of boundaries are shown in Fig. 5.

We parametrize the lattices in our family by a parameter d which will be equal to the code distance of the three codes supported on a particular lattice. We specify the structure of a distance d lattice by dividing it into 2D layers which are parallel to the half-cuboctahedra boundaries. There are two types of layers in this division, which we call "checkerboard layers" and "diamond layers," due to their appearance. Figure 6 shows the structure of the two types of layers in the $d = 3$ lattice and the $d = 4$ lattice. In a distance d lattice, there are d checkerboard layers and $d - 1$ diamond layers and the two types of layers alternate. The half-cuboctahedra boundaries are themselves checkerboard layers. Layers directly above and below each other are connected by edges as can be seen in Fig. 5.

2. Code structure

In this section, we specify the structure of the three 3D surface codes defined on the same distance d rectified cubic lattice. We place three qubits at each vertex of the lattice (one qubit per code). Each checkerboard layer in the lattice has d^2 vertices each and each diamond layer has $2d(d-1)$ vertices. Therefore, for a distance d lattice, the number of physical qubits in each code is

$$\begin{aligned} n &= d^3 + 2d(d-1)^2 \\ &= 3d^3 - 4d^2 + 2d. \end{aligned} \tag{1}$$

We label each code $\mathcal{SC}_c$ with the color of its X stabilizers. $\mathcal{SC}_c$ has X stabilizers associated with c cells and Z stabilizers associated with $c'c''$ faces. The following Table and Fig. 7 detail the stabilizers of the three codes supported on a rectified cubic lattice:

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

We also associate colors with the boundaries of our rectified cubic lattices. A c boundary corresponds to a rough boundary in $\mathcal{SC}_c$ and smooth boundaries in $\mathcal{SC}_{c'}$ and $\mathcal{SC}_{c''}$. In Sec. II, we defined rough and smooth boundaries in terms of quasiparticle condensation. For regular lattices like the ones we consider, we can be more specific about the structure of the boundaries. In a 3D surface code defined on a cubic lattice in the Kitaev picture, each qubit in the bulk is a member of two X stabilizers and four Z stabilizers. Similarly, in a 3D surface code defined on a tetrahedral-octahedral lattice in the Kitaev picture, each qubit in the bulk is a member of two X stabilizers and four Z stabilizers. In each of these lattices, the qubits on the rough boundaries are members of a single X stabilizer and the qubits on the smooth boundaries are members of between one and three Z stabilizers (i.e., fewer than four). We note that the parts of lattices at which two boundaries meet are part of both boundaries.

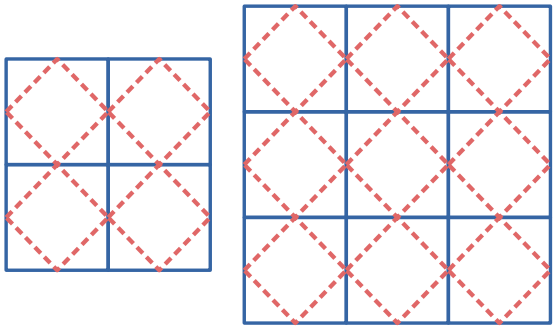


FIG. 6. The two types of layers in a $d = 3$ rectified cubic lattice (left) and a $d = 4$ rectified cubic lattice (right). Checkerboard layers (continuous blue lines) are layers which slice cuboctahedra in half and diamond layers (dashed red lines) are layers which slice octahedra in half.

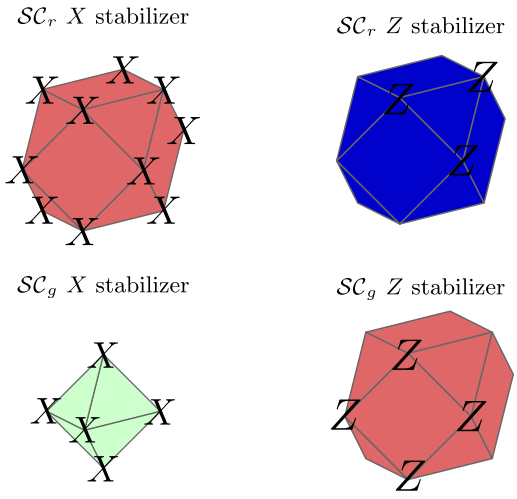


FIG. 7. The stabilizers of $\mathcal{SC}_r$ and $\mathcal{SC}_g$. The $\mathcal{SC}_b$ stabilizers are identical to those of $\mathcal{SC}_r$ except with red (medium gray) and blue (dark gray) interchanged.

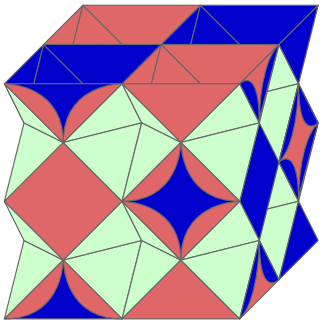


FIG. 8. The additional stabilizers required such that the codes in our family of stacked 3D surface codes have the correct boundaries. First, consider the full-cuboctahedra boundary facing us. We associate additional $\mathcal{SC}_r$ X stabilizers with the faces of the b cuboctahedra on this boundary [blue (dark gray) faces]. In addition, we associate additional $\mathcal{SC}_b$ Z stabilizers with some of the edges of these blue (dark gray) faces [red (medium gray) circular segments]. These edges would have been part of rg faces if not for the boundaries. In effect, we have added a multiple 2D flattenings of r cuboctahedra to the lattice. The edges of these 2D flattenings are themselves 1D flattenings of rg faces ($\mathcal{SC}_b$ Z stabilizers). We add analogous stabilizers to the back boundary. With these additional stabilizers, the front and back boundaries are valid b boundaries. Next, consider the left and right boundaries in the figure. We associate additional $\mathcal{SC}_b$ X stabilizers with the faces of the r cuboctahedra [red (medium gray) faces] on these boundaries. We also associate additional $\mathcal{SC}_r$ Z stabilizers with some of the edges of these faces [blue (dark gray) circular segments]. With these additional stabilizers, the left and right boundaries are valid r boundaries.

For our family of stacked 3D surface codes to have a transversal CCZ gate (see Sec. V), we need to have two boundaries of each color and we need opposite boundaries to have the same color. The half-cuboctahedra boundaries of the distance d rectified cubic lattices we detailed in the previous section are 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. We need to break the symmetry between the four full-cuboctahedra boundaries to turn them into valid r boundaries and b boundaries. We break the symmetry by adding additional low-weight stabilizers to the full-cuboctahedra boundaries. These stabilizers are analogous to the weight-two stabilizers on the boundaries of the [[9,1,3]] 2D surface code shown in Fig. 1. In Fig. 8 we show the additional stabilizers we add to $\mathcal{SC}_r$ and $\mathcal{SC}_b$ to turn the full-cuboctahedra boundaries into r boundaries and b boundaries.

With the additional stabilizers shown in Fig. 8, we claim that the three codes have the correct structure on the boundaries of the lattice. That is, the c boundaries are rough boundaries in $\mathcal{SC}_c$ and smooth boundaries in $\mathcal{SC}_{c'}$ and $\mathcal{SC}_{c''}$. First consider the g boundaries (top and bottom boundaries in Fig. 8). Each vertex on the g 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 the g boundary meets the r boundaries and b boundaries. The g boundary is, therefore, a rough boundary in $\mathcal{SC}_g$. Each vertex on the g boundaries is a member of two r cuboctahedra (including 2D flattenings shown in Fig. 8) and two b cuboctahedra (including 2D flattenings), except where the g boundaries meet an r boundary or a b boundary, respectively. The vertices on the g boundaries are all members of fewer than four rg faces (including 1D flattenings shown in Fig. 8) and fewer than four bg faces (including 1D flattenings). Therefore, the g boundaries are smooth boundaries in $\mathcal{SC}_b$ and $\mathcal{SC}_r$.

The next pair of boundaries we consider are the b boundaries. Due to the additional stabilizers shown in Fig. 8, each vertex on the b boundaries is a member of two r cuboctahedra (including 2D flattenings) and two g octahedra, except where the b boundaries meet the r boundaries and g boundaries, respectively. However, each vertex on the b boundaries is a member of a single b cuboctahedron. Every vertex on the b 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 which are also on r boundaries or g boundaries. Therefore, the b boundaries are rough boundaries in $\mathcal{SC}_b$ and smooth boundaries in $\mathcal{SC}_r$ and $\mathcal{SC}_g$, as required. The argument for r boundaries is identical to the argument for b boundaries, except with r and b exchanged. In Appendix B, we describe an alternative family of stacked 3D surface codes which are supported on rectified cubic lattices with different boundaries to the ones we have just described.

Next, we show that each of the three codes has one encoded logical qubit. The number of encoded qubits in a stabilizer code is equal to the number of physical qubits minus the number of stabilizer generators. So, we need to count the number of stabilizer generators in each of the three codes. We begin with $\mathcal{SC}_g$. In this code, X stabilizers are associated with g cells (octahedra) and Z stabilizers are associated with rb faces. Consider the top g boundary of a distance d lattice oriented the same way as the $d = 3$ lattice in Fig. 8. This boundary has the structure of a checkerboard layer and each vertex on this boundary is a member of a single (complete or incomplete) octahedron. Checkerboard layers have d^2 vertices so we have d^2 octahedra which are situated directly below the top boundary. Every other checkerboard layer (except the bottom layer) also has d^2 octahedra situated below it. There are d checkerboard 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 X stabilizer generators in $\mathcal{SC}_g$ is

$$\left|S_X^{(g)}\right| = d^2(d-1). \tag{2}$$

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 which are parallel to g boundaries, and faces which 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 checkerboard layer cuts through $2d(d-1)$ rb faces which are

parallel to the r boundaries or the b boundaries. There are d checkerboard 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 stabilizer generators for every cuboctahedron in the lattice to get a set of independent Z generators. Each checkerboard layer has $(d-1)^2$ cuboctahedra and there are d checkerboard layers, so in total we have $d(d-1)^2$ cuboctahedra in a distance d lattice. Therefore, the total number of Z stabilizer generators in $\mathcal{SC}_g$ is

$$\left|S_Z^{(g)}\right| = (d-1)(2d^2-d+1). \tag{3}$$

The total number of stabilizer generators in $\mathcal{SC}_g$ is therefore

$$\begin{aligned}\left|S_X^{(g)}\right| + \left|S_Z^{(g)}\right| &= (d-1)(3d^2-d+1)\\ &= 3d^3-4d^2+2d-1.\end{aligned} \tag{4}$$

By comparing Eqs. (4) and (1), we see that $\mathcal{SC}_g$ has $n-1$ 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 stabilizers of this code are associated with b cells (including the 2D flattenings) and the Z stabilizers are associated with rg faces (including the 1D flattenings). First, we count the X stabilizers of $\mathcal{SC}_b$. Consider the checkerboard layers parallel to the g boundaries. Each checkerboard layer has $(d-1)^2$ cuboctahedra (half of which are r and half of which are b). There are d checkerboard 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 Fig. 8). 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 total number of X-stabilizer generators in $\mathcal{SC}_b$ is

$$\left|S_X^{(b)}\right| = \frac{(d-1)}{2}(d^2+d). \tag{5}$$

Next, we count the Z-stabilizer generators of $\mathcal{SC}_b$. The Z stabilizers of $\mathcal{SC}_b$ 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 have four rg faces. The checkerboard layers which 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 checkerboard 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 Fig. 8, we also have $\mathcal{SC}_b$ Z stabilizers which are associated with the edges of the faces which 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 $2(d-1)$ triangular faces and $(d-1)(d-2)$ square faces on the b boundaries which belong to b cuboctahedra in a distance d lattice. Therefore, the total number of independent weight-two Z stabilizers in $\mathcal{SC}_b$ is $(d-1)(3d-2)$.

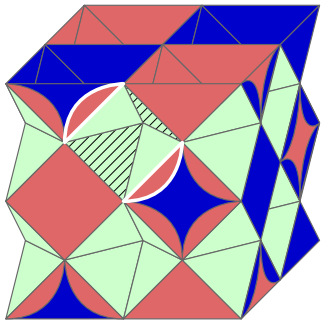


FIG. 9. Redundant Z stabilizers in $\mathcal{SC}_b$. We can construct the identity by multiplying the Z stabilizers associated with the rg faces and edges of half octahedra on the b boundaries. We have highlighted one such collection of faces (hatched green triangles) and circular segments (red faces with white edges).

Some of the Z stabilizers we have counted so far are not independent. Consider a complete 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 lose a single Z stabilizer from the list of stabilizer generators for each complete octahedron and r cuboctahedron. Every checkerboard 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 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 Fig. 9. 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 stabilizer generators. The total number of Z-stabilizer generators in $\mathcal{SC}_b$ is therefore

$$\begin{aligned}\left|S_Z^{(b)}\right| &= \frac{(d-1)}{2}[8(d-1)^2+6d-4-3d^2+7d-2]\\ &= \frac{d-1}{2}(5d^2-3d+2).\end{aligned} \tag{6}$$

The total number of stabilizer generators in $\mathcal{SC}_b$ is

$$\begin{aligned}\left|S_X^{(b)}\right| + \left|S_Z^{(b)}\right| &= (d-1)(3d^2-d+1)\\ &= 3d^3-4d^2+2d-1.\end{aligned} \tag{7}$$

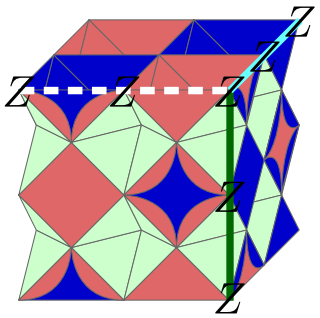


FIG. 10. Canonical $\overline{Z}_r$ (dashed white line), $\overline{Z}_g$ [continuous green (dark gray) line], and $\overline{Z}_b$ [continuous blue (light gray) line] operators. The canonical $\overline{X}_c$ operators act on every qubit on one of the c boundaries.

By comparing Eqs. (7) and (1), we see that $\mathcal{SC}_b$ has has $n-1$ 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. The argument showing this is identical to the argument for $\mathcal{SC}_b$, except with r and b swapped everywhere. In Appendix A, we list the stabilizer generators of the three codes supported on a $d=2$ rectified cubic lattice.

3. Logical operators

To finish our discussion of rectified cubic codes, we detail the logical operators of the three surface codes supported on a distance d rectified cubic lattice. $\overline{Z}_c$ 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 which are members of both boundaries. Figure 10 shows example canonical $\overline{Z}_c$ operators for the three codes in a single stack. These canonical $\overline{Z}_c$ operators are weight d, where d is the code distance that parametrizes the lattice. We define the canonical $\overline{X}_c$ operators as membranes of X operators which 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 $d^2+(d-1)^2$.

B. Other rectified picture lattices

It is natural to wonder whether the rectified cubic lattice is the only lattice which supports three 3D surface codes in the rectified picture. We say a lattice supports three 3D surface codes in the rectified picture if we can partition the cells and faces of the lattice into three sets such that we can define a valid 3D surface code for each set (with X stabilizers associated with cells and Z stabilizers associated with faces). In the 2D case, there are many lattices which support two surface codes in the rotated picture. Indeed, any four-valent lattice will work [37].

For 3D lattices, the situation is more complex. To make the analysis easier, we consider rectified lattices without boundaries. To support three 3D surface codes, a rectified picture lattice must satisfy the following conditions:

(1) The cells must be 3-colorable.

(2) Each vertex must be part of exactly two cells of each color.

(3) Each vertex must be part of three or more faces of each color.

(4) All c cells and faces which are not part of c cells must have an even number of vertices in common.

Condition 1 allows us to assign colors to the cells and faces in a consistent way. We assign each face the colors of the two cells of which it is a member. As with rectified cubic codes, we assign each surface code $\mathcal{SC}_c$ a color. In $\mathcal{SC}_c$, we associate X stabilizers with c cells and Z stabilizers with $c'c''$ faces. Condition 2 ensures that each qubit is acted upon nontrivially by exactly two X stabilizers in each code. This is necessary because in the Kitaev picture (primal lattice) qubits are associated with edges and X stabilizers with vertices. Condition 3 ensures that each qubit is acted upon nontrivially by three or more Z stabilizers in each code. This condition is necessary to ensure that the m quasiparticles are 1D objects, as required in 3D surface codes. Finally, condition 4 ensures that the X and Z stabilizers in each code commute. In addition, we note that condition 4 implies Lemma 1. This means that as long as the three 3D surface codes have canonical logical operators which overlap as described in Fig. 16, they will have a transversal CCZ gate. The only semiregular (vertex-transitive) 3D lattice we have found which satisfies the above conditions is the rectified cubic lattice. However, it is likely that other less regular lattices exist which satisfy the conditions.

If we relax condition 1, we can find regular rectified picture lattices which support more than three 3D surface codes. Instead of insisting on 3-colorability, we allow the cells of the lattice to be 4-colorable. For example, consider the cubic lattice. We can color the cells of this lattice with four colors such that each cube has the same color as the cubes with which it shares exactly one vertex (see Fig. 11). With this coloring, the cubic lattice supports four 3D surface codes. We choose the four colors $\{r, g, b, y\}$. The four codes have the following stabilizer groups:

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

The idea of defining a 3D surface code on the cubic lattice in this way is due to Kubica [40]. However, he did not consider multiple surface codes defined on the same lattice. We have not constructed a family of codes supported on cubic lattices with boundaries, but this may be possible.

Remarkably, the cubic lattice surface codes we defined above are a gauge choice of the 3D Bacon-Shor code [41], a

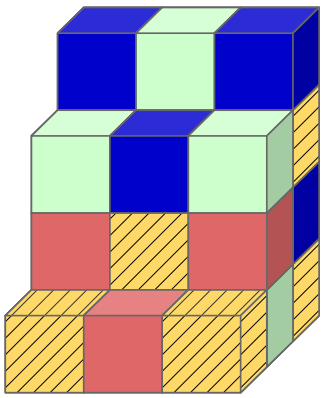


FIG. 11. A cubic lattice colored with four colors [blue (dark gray), red (medium gray), green (light gray), and yellow (hatched)]. Cubes which share exactly one vertex have the same color.

well-known subsystem code [42]. A similar result is widely known in the 2D case (see, e.g., [43]). Subsystem codes are quantum error-correcting codes where the encoded qubits separate into two sets: gauge qubits and logical qubits. We only use the logical qubits to encode information, but the gauge qubits give subsystem codes additional structure which is not present in stabilizer codes. A subsystem code is defined by its gauge group $\mathcal{G}$, a subgroup of the Pauli group. The stabilizer group of the subsystem code is the center of the gauge group $\mathcal{S} = \mathcal{Z}(\mathcal{G})$. The nontrivial logical operators of a subsystem code are the elements of the Pauli group which commute with all the stabilizers but are not in the gauge group. In the 3D Bacon-Shor code, we place qubits on the vertices of a cubic lattice. The gauge group $\mathcal{G}$ is generated by XX and ZZ operators associated with the edges of the lattice. The X-type gauge generators are associated with edges in the i and j directions. Similarly, the Z-type gauge generators are associated with edges in the j and k directions. The stabilizer group contains "nearest-plane" operators. That is, the X-type stabilizers consist of X operators acting on all the qubits in two jk planes which are next to each other in the i direction. Similarly, the Z-type stabilizers consist of Z operators acting on all the qubits in two ij planes which are next to each other in the k direction.

A stabilizer code defined by the stabilizer group $\mathcal{S}$ is a gauge choice of a subsystem code defined by the gauge group $\mathcal{G}_1$ and stabilizer group $\mathcal{S}_1$ if the following inclusions hold [17,44]:

$$\mathcal{S}_1 \subseteq \mathcal{S} \subseteq \mathcal{G}_1. \tag{8}$$

Consider $\mathcal{SC}_r$ as defined above. The X stabilizers of $\mathcal{SC}_r$ are associated with r cubes. Clearly, we can construct these cube operators from X gauge operators associated with the edges in the i and j directions. Similarly, we can construct the Z stabilizers of $\mathcal{SC}_r$ (bg, by, and gy faces) from Z gauge operators associated with the edges in the j and k directions. In addition, the stabilizer generators of the 3D Bacon-Shor code can be constructed from the stabilizer generators of $\mathcal{SC}_r$. We can construct any X "nearest-plane" operator from X r-cube operators and we can construct any Z "nearest-plane operator" from Z bg, by, and gy face operators. The same is true for all the other 3D surface codes defined above by symmetry. Therefore, 3D surface codes defined on the cubic lattice (in the rectified picture) are particular gauge choices of the 3D Bacon-Shor code.

IV. CONCATENATION TRANSFORMATION

In this section, we show how to transform three 3D surface codes into a 3D color code using code concatenation. Color codes are a family of topological codes introduced by Bombín and Martin-Delgado [23,45]. 3D color codes are defined on weakly four-valent, 4-colorable lattices. In a weakly four-valent lattice, all vertices are four-valent except for vertices on the boundaries. In a 3D color code, we place qubits on the vertices of the lattice, we associate X stabilizers with the cells of the lattice, and we associate Z stabilizers with the faces of the lattice. This makes 3D color codes very similar to 3D surface codes in the rectified picture. In fact, 3D color codes and stacks of three 3D surface codes are equivalent up to local Clifford unitaries, as shown by Kubica *et al.* [21]. This result is a special case of their more general result which states that D copies of a D-dimensional surface code are local Clifford equivalent to a single D-dimensional color code. The surprisingly close relationship between color codes and surface code has also been explored in a number of other works [22,46–49].

Criger and Terhal gave an explicit construction of the local Clifford unitaries required to transform two 2D surface codes into a single 2D color code [27]. Their construction is remarkably simple: it consists of encoding pairs of qubits (one from each 2D surface code) in the [[4,2,2]] error-detecting code. This code can be viewed as a 2D color code defined on a single square. It has two stabilizers $X^{\otimes 4}$ and $Z^{\otimes 4}$ and weight-two logical operators supported on the sides of the square. The [[4,2,2]] code has a transversal CZ gate implemented using $S = \mathrm{diag}(1, i)$ and $S^\dagger$ gates. We can generalize this code-concatenation transformation to 3D. Instead of a [[4,2,2]] code we use an [[8,3,2]] code. We can view this code as a small 3D color code defined on a cube [as shown in Fig. 12(a)]. It has an X stabilizer acting on all of the qubits and Z stabilizers associated with the faces of the cube. Only four of the Z face stabilizers are independent, so this code has three encoded qubits. Logical $\overline{X}$ operators are membranes of X operators which act on four qubits on the same face (opposite faces support $\overline{X}$ operators which act on the same encoded qubit). $\overline{Z}$ operators are strings of Z operators that act on the qubits at the end points of edges linking the faces which support the corresponding $\overline{X}$ operators. The vertices of a cube are 2-colorable, i.e., we can assign each vertex a color such that no vertices which share an edge have the same color. We can implement a transversal CCZ in the [[8,3,2]] code by applying $T = \mathrm{diag}(1, e^{i\pi/4})$ gates to qubits on vertices of one color and $T^\dagger$ gates to the qubits on the vertices of the other color. This fact can be verified by computing the action of T and $T^\dagger$ on the codeword kets.

In a 3D color code, we assign faces the colors of the cells they are members of. For example, a face which is a member of a c cell and a c' cell is a cc' face. Due to the 4-colorability of the color-code lattice, each cell's faces are

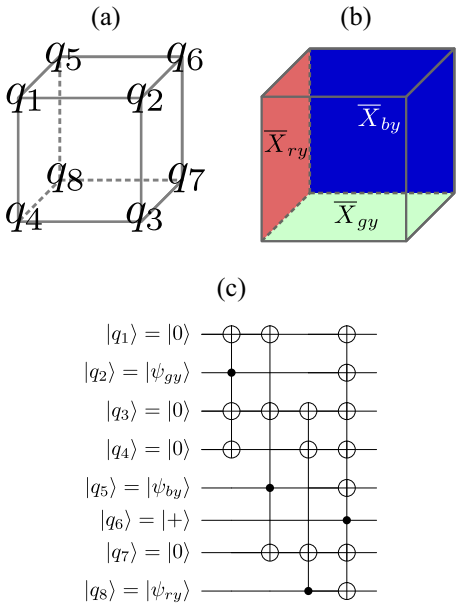


FIG. 12. The [[8,3,2]] color code. (a) Our labeling of the qubits. The X stabilizer acts on all the qubits (it is a cell operator) and the Z stabilizers act on qubits which are members of the same face. (b) In a larger 3D color code, the [[8,3,2]] cube would have an assigned color (say y) and its faces would be 3-colorable ($c \in \{ry, by, gy\}$). $\overline{X}_{cy}$ is supported either of the cy faces (opposite faces have the same color) and $\overline{Z}_{cy}$ is supported on an edge that links the cy faces. (c) The encoding circuit for the [[8,3,2]] code. Encoded $\overline{X}_{cy}$ operators [shown in (b)] act on the encoded qubit $|\overline{\psi}_{cy}\rangle$. We derived this circuit using the method given in [34].

3-colorable. Consider a color code lattice where cells assigned colors from the set $\{r, g, b, y\}$. We can view the [[8,3,2]] code as a cell of this lattice. Assume that it is a y cell. Then, its faces are colored ry, by, and gy. We use these colors to index the logical operators of the [[8,3,2]] code. That is, the logical $\overline{X}$ operators which act on the cy faces are denoted by $\overline{X}_{cy}$. These operators are shown in Fig. 12(b). We denote the corresponding $\overline{Z}$ operators as $\overline{Z}_{cy}$.

We can now detail the concatenation transformation which maps a stack of three 3D surface codes to a single 3D color code. Consider a rectified cubic code stack with code distance d. To transform the three codes in the stack, we encode the three qubits at every vertex in [[8,3,2]] codes. An encoding circuit for the [[8,3,2]] code is shown in Fig. 12(c). Figure 13

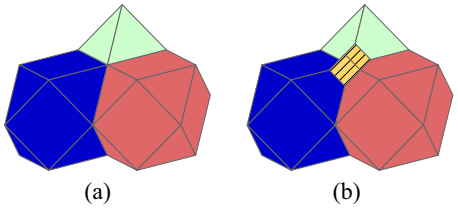


FIG. 13. An [[8,3,2]] concatenation transformation of a single vertex in a stack of 3D surface codes. (a) The initial rectified cubic lattice. (b) We encode the three qubits at the vertex where the three different cells meet in an [[8,3,2]] color code. This corresponds to replacing the vertex with a cube [yellow (hatched) cell].

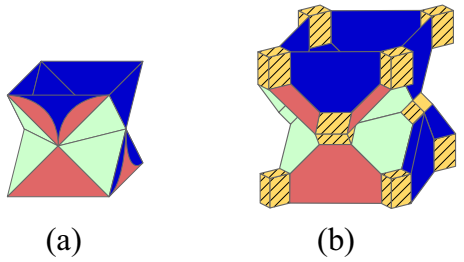


FIG. 14. Transforming a stack of three 3D surface codes into a single 3D color code by concatenating with the [[8,3,2]] color code. Each vertex the $d = 2$ rectified cubic lattice (a) is transformed as shown in Fig. 13. This transforms the rectified cubic lattice into a cantitruncated cubic lattice (b). This lattice supports a $d = 4$ color code with three encoded qubits. The top and bottom boundaries of the surface-code stack are g boundaries, the left and right boundaries are r boundaries, and the front and back boundaries are b boundaries. In a color code, a c boundary is a boundary which has no c cells adjacent to it. By inspecting (b), we see that c boundaries in the surface-code stack become c boundaries in the color code.

shows the [[8,3,2]] concatenation transformation applied to a single vertex. Applied to a whole lattice, concatenation with the [[8,3,2]] code transforms cuboctahedra into truncated cuboctahedra, octahedra into truncated octahedra, and vertices into cubes. Globally, this transforms the rectified cubic lattice into a cantitruncated cubic lattice. Two truncated cuboctahedra, one truncated octahedron, and one cube meet at each vertex of a cantitruncated cubic lattice. Figure 14 shows how a $d = 2$ rectified cubic lattice transforms under the [[8,3,2]] concatenation transformation.

The colors we assigned to the encoded qubits of the [[8,3,2]] codes tell us how to encode the three qubits at every vertex of the rectified cubic lattice. We encode the physical qubits from $\mathcal{SC}_c$ as the cy qubits of the [[8,3,2]] codes [see Fig. 12(c)]. This ensures that $\mathcal{SC}_c$ X (Z) stabilizers associated with c cells (cc' faces) are mapped to X (Z) stabilizers associated with c cells (cc' faces) in the color code. In each 3D surface code we have n qubits and $n-1$ independent stabilizer generators. In the color code we have $8n$ qubits and we inherit $3(n-1)$ stabilizer generators. We also have five independent stabilizer generators for each cube (one X stabilizer and four Z stabilizers). So, in total we have $5n + 3n - 3 = 8n - 3$ independent stabilizer generators in the 3D color code. The 3D color code therefore encodes three logical qubits. The color code inherits the boundary structure of the stack of 3D surface codes. In a color code, a boundary has the color c, if no c cells are present on it. As shown in Fig. 14, the c boundaries of the rectified cubic lattice become c boundaries in the color code.

As with surface codes, we interpret unsatisfied color code stabilizers as quasiparticles. For each color c in a 3D color code, we have quasiparticles e_c and m_c. In the stack of surface codes, each code $\mathcal{SC}_c$ has quasiparticles e_c and m_c. The quasiparticles of the three surface codes are mapped directly to quasiparticles in the color code. For any color $c \in \{r, g, b\}$, the e_c (m_c) quasiparticles in our three 3D surface codes are mapped to e_c (m_c) quasiparticles in the color code because c-cell (cc'-face) stabilizers in the surface codes are

mapped to c-cell (cc'-face) stabilizers in the color code. This leaves the y quasiparticles in the color code unaccounted for. However, this is not important as the y quasiparticles are not independent. We can always construct y quasiparticles from combinations of r, g, and b quasiparticles [23].

The logical operators of the color code have the same structure as the logical operators of the three surface codes. In the color code, $\overline{Z}_c$ operators are strings of Z operators from one c boundary to the other and $\overline{X}_c$ operators are membranes of X operators with boundaries that span the c' and c'' boundaries. As the concatenation transformation maps the c boundaries of the rectified cubic codes to c boundaries in the color code, the structure of the logical operators is preserved by the mapping.

V. A UNIVERSAL GATE SET IN 3D SURFACE CODES

In this section, we prove that CCZ and CZ are transversal in rectified cubic codes and we show how to implement a universal gate set in these codes. We note that CZ is also transversal in 2D surface codes. This fact can be easily understood in the rotated picture, as we explain in Appendix D.

An important concept in our proofs is the overlap of logical operators (including stabilizers). Given two or three logical operators, each of which acts on a different code in a rectified cubic stack, we define the overlap of these operators as the vertices where all the operators act nontrivially. Before proceeding to the main proofs, we need the following lemma about rectified cubic codes.

Lemma 1. The overlap of any two X stabilizers from two different codes in a rectified cubic code stack is equal to the nontrivial support of a Z stabilizer from the third code.

In other words, the set of vertices at which both X stabilizers act nontrivially are equal to the support of some Z stabilizer in the third code.

Proof. We initially restrict our attention to the bulk of the lattice. Let us consider X stabilizer generators from $\mathcal{SC}_r$ (r cells) and $\mathcal{SC}_g$ (g cells). We denote the X and Z stabilizers of $\mathcal{SC}_c$ as S_c^x and S_c^z, respectively. Clearly, S_r^x generators and S_g^x generators overlap on rg faces (S_b^z operators) in the bulk. This is also true for the other two color combinations.

On the boundaries, S_r^x generators and S_b^x operators overlap on rb faces. This can be seen by inspecting, e.g., Fig. 8. Some S_r^x generators and S_g^x generators overlap on edges. However, in all these cases, a S_b^z operator is supported on the overlap edge. An example of this is highlighted in Fig. 15. Similarly, S_b^x and S_g^x generators on the boundaries can overlap on edges. But, all these edges have an associated S_r^z operator.

We have shown that all pairs of X stabilizer generators from two different codes in the stack overlap on faces or edges which support Z stabilizers in the third code. As every X stabilizer is a product of stabilizer generators, any pair of X stabilizers from two different codes have overlap equal to the support of a Z stabilizer from the third code. ■

A. Transversal CCZ

We now prove that CCZ is transversal for stacks of rectified cubic codes. We first write the surface-code kets in a form inspired by a proof in [44]. Let H_c^x be the (classical) parity check matrix of the X stabilizers of $\mathcal{SC}_c$. That is, H_c^x is an $m \times n$ binary matrix with m equal to the number of X-stabilizer generators in $\mathcal{SC}_c$ and n equal to the number of physical qubits in the code. Each row of H_c^x has a 1 at column j if the stabilizer generator corresponding to that row acts nontrivially on qubit q_j. If the stabilizer generator acts trivially, then the entry is equal to zero. Now, let G_c^0 be the linear span the rows of H_c^x. For each code, we choose a canonical $\overline{X}_c$ operator which acts on one of the c boundaries of the lattice. Let X_c be an n-bit binary vector describing the support of $\overline{X}_c$. That is, X_c has a one at position j if $\overline{X}_c$ acts nontrivially on qubit q_j, with all other entries in X_c equal to zero. Let G_c^1 be the coset $\{X_c + g : g \in G_c^0\}$. With these definitions we can write the encoded state of $\mathcal{SC}_c$ as follows:

$$|\overline{\alpha}\rangle_c = \frac{1}{\sqrt{|G_c^\alpha|}} \sum_{g \in G_c^\alpha} |g\rangle_c \,, \tag{9}$$

where $|G_c^\alpha|$ is the number of elements in G_c^α and $\alpha \in \{0, 1\}$.

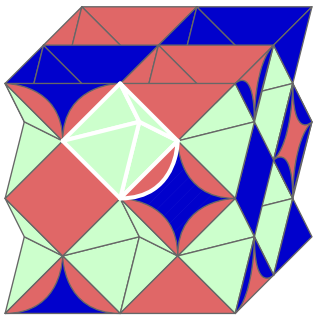


FIG. 15. The overlap of a S_g^x generator [green (light gray) octahedron with white edges] and a S_r^x generator (hatched blue face) is equal to an edge [red (medium gray) circular segment with white edges]. This edge has an associated S_b^z operator, as explained in Fig. 8.

To show that CCZ is transversal for stacked 3D surface codes, we need the following lemma.

Lemma 2. Given a finite set of k binary vectors $\{a_j\}$ with the same length, 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 C.

Theorem 3. CCZ is transversal in rectified cubic codes.

Proof. Define $\overline{\mathrm{CCZ}} = \mathrm{CCZ}^{\otimes n}$, 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

$$|\overline{\alpha\beta\gamma}\rangle_{rgb} = \sum_{t \in G_r^\alpha u \in G_g^\beta v \in G_b^\gamma} |t\rangle_r |u\rangle_g |v\rangle_b \,, \tag{10}$$

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

$$\begin{aligned} \overline{\mathrm{CCZ}} |\overline{\alpha\beta\gamma}\rangle_{rgb} &= \sum_{t \in G_r^\alpha u \in G_g^\beta v \in G_b^\gamma} \mathrm{CCZ}^{\otimes n} |t\rangle_r |u\rangle_g |v\rangle_b \\ &= \sum_{t \in G_r^\alpha u \in G_g^\beta v \in G_b^\gamma} (-1)^{|t \circ u \circ v|} |t\rangle_r |u\rangle_g |v\rangle_b \,, \end{aligned} \tag{11}$$

where $u \circ v$ denotes the bitwise binary product between u and v and $|t|$ denotes the Hamming weight of t.

We now calculate $(-1)^{|t\circ u\circ v|}$ for each encoded computational basis state. We can expand $t \circ u \circ v$ as follows

$$\begin{aligned} t \circ u \circ v &= (\alpha X_r + t') \circ (\beta X_g + u') \circ (\gamma X_b + v') \\ &= \alpha\beta\gamma (X_r \circ X_g \circ X_b) + \alpha\beta (X_r \circ X_g \circ v') \\ &\quad + \alpha\gamma (X_r \circ X_b \circ u') + \beta\gamma (X_g \circ X_b \circ t') \\ &\quad + \alpha (X_r \circ u' \circ v') + \beta (X_g \circ t' \circ v') \\ &\quad + \gamma (X_b \circ t' \circ u') + (t' \circ u' \circ v'), \end{aligned} \tag{12}$$

where $t' \in G_0^r$, $u' \in G_0^g$, and $v' \in G_0^b$.

First, we consider the term $(t' \circ u' \circ v')$, which corresponds to the state $|\overline{000}\rangle$. We can find the Hamming weight of this term by considering the support of the stabilizers which correspond to t', u', and v'. The t' vectors correspond to the X stabilizers of $\mathcal{SC}_r$ (S_r^x), the u' vectors correspond to the X stabilizers of $\mathcal{SC}_g$ (S_g^x), and the v' vectors correspond to the X stabilizers of $\mathcal{SC}_b$ (S_b^x). The Hamming weight of the product $t' \circ u' \circ v'$ will be equal to the number of vertices in the lattice where the three X stabilizers act nontrivially on the physical qubits of their respective codes. In other words, it will be equal to the overlap of the three operators.

By Lemma 1, any S_r^x operator and any S_g^x operator have overlap equal to the support of a S_b^z operator ($\mathcal{SC}_b$ Z stabilizer). As the stabilizers of $\mathcal{SC}_b$ commute, the overlap of any S_r^x, S_g^x, and S_b^x is always even. Hence, $|t' \circ u' \circ v'| = 0 \mod 2$ for all t', u', and v' and $(-1)^{|t\circ u\circ v|} = 1$ for $|\overline{000}\rangle$.

Next, we consider the exponent for $|\overline{001}\rangle$ which is equal to $(X_b \circ t' \circ u') + (t' \circ u' \circ v')$. Thanks to Lemma 2 we only need to show that $(X_b \circ t' \circ u')$ has even Hamming weight to show that the sum has even Hamming weight. We need to calculate the overlap of the $\overline{X}_b$ operator on the b boundary (corresponding to the X_b vector) with any S_r^x and S_g^x. By Lemma 1, any S_r^x and S_g^x overlap on a collection of vertices which has the same support (in terms of vertices) as a S_b^z operator. Logical operators and stabilizers commute, so the overlap of $\overline{X}_b$ with any S_r^x and S_g^x is even. This implies that $|(X_b \circ t' \circ u')| = 0 \mod 2$ for every t' and u'. All the other terms in Eq. (12) with one X_c term have even Hamming weight by the same argument. Therefore, $(-1)^{|t\circ u\circ v|} = 1$ for $|\overline{100}\rangle$, $|\overline{010}\rangle$, and $|\overline{001}\rangle$.

The next computational basis state we consider is $|\overline{110}\rangle$. The exponent for this state is $(X_r \circ X_g \circ v') + (X_r \circ u' \circ v') + (X_g \circ t' \circ v') + (t' \circ u' \circ v')$. To show that this expression has even Hamming weight, we only need to show that $(X_r \circ X_g \circ v')$ has even Hamming weight due to Lemma 2. To find the Hamming weight of this term, we need to find the overlap of $\overline{X}_r$, $\overline{X}_g$, and any S_b^x operator. $\overline{X}_r$ has nontrivial support on an r boundary and $\overline{X}_g$ has nontrivial support on a g boundary. These two operators overlap on a line where the r boundary and the g boundary meet (shown in Fig. 16). This line is a string from one b boundary to the other b boundary, i.e., it has the same support as a $\overline{Z}_b$ operator. Logical operators and stabilizers commute so $\overline{X}_r$, $\overline{X}_g$, and any S_b^x have even overlap. This proves that $|(X_r \circ X_g \circ v')| = 0 \mod 2$ for all v'. All the other terms in the Eq. (12) expansion with two X_c terms have even Hamming weight by the same argument. Hence, $(-1)^{|t\circ u\circ v|} = 1$ for $|\overline{110}\rangle$, $|\overline{101}\rangle$, and $|\overline{011}\rangle$.

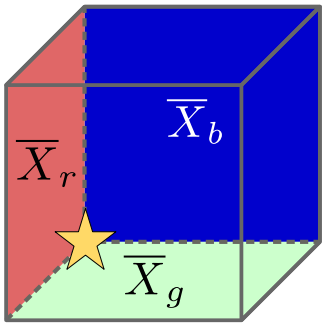


FIG. 16. The overlap of the $\overline{X}_c$ operators which lie on the boundaries. We see that any $\overline{X}_c$ and $\overline{X}_{c'}$ overlap on a $\overline{Z}_{c''}$ path (a string from one c'' boundary to the other). The three $\overline{X}_c$ operators overlap at a single vertex (denoted by a star).

Finally, for the state $|\overline{111}\rangle$ we must consider the entire expansion in Eq. (12). Due to the previous calculations in this proof and Lemma 2, the parity of this exponent is determined by $(X_r \circ X_g \circ X_b)$. This term has Hamming weight equal to the number of lattice collisions between $\overline{X}_r$, $\overline{X}_g$, and $\overline{X}_b$. As these three operators are defined on r, g, and b boundaries, respectively, they have a single lattice collision on one corner of the lattice (shown in Fig. 16). Therefore, $|(X_r \circ X_g \circ X_b)| = 1$ which implies that $(-1)^{|t\circ u\circ v|} = -1$ for $|\overline{111}\rangle$.

We have shown that $\overline{\mathrm{CCZ}}$ has has the correct action on the computational basis states, namely,

$$\overline{\mathrm{CCZ}}\,|\overline{\alpha\beta\gamma}\rangle = \begin{cases} -|\overline{\alpha\beta\gamma}\rangle & \alpha = \beta = \gamma = 1, \\ |\overline{\alpha\beta\gamma}\rangle & \text{else.} \end{cases} \tag{13}$$

■

B. Transversal CZ

The transversality of CZ in stacked 3D surface codes follows from the structure of the CCZ and X operators. Consider three codes, each encoding one logical qubit, labeled with the labels i, j, and k. We assume that $\overline{\mathrm{CCZ}}_{ijk}$ is a transversal gate acting as a tensor product of CCZ gates at the level of the physical qubits. In addition, we assume that each $\overline{X}$ gate acts as a tensor product of X gates at the level of the physical qubits. The group commutator of two operators A and B is defined as $K[A, B] = ABA^\dagger B^\dagger$. One can easily verify that $K[\mathrm{CCZ}_{ijk}, X_k] = \mathrm{CZ}_{ij}$. Therefore, we can implement a transversal $\overline{\mathrm{CZ}}_{ij}$ gate by applying the sequence of logical operators $K[\overline{\mathrm{CCZ}}_{ijk}, \overline{X}_k]$. If we think at the level of the physical qubits, this operator simplifies. All triples of qubits outside the support of $\overline{X}_k$ are acted upon by $\mathrm{CCZ}_{ijk}\mathrm{CCZ}_{ijk}^\dagger = I$ and triples of qubits in the support of $\overline{X}_k$ are acted upon by $K[\mathrm{CCZ}_{ijk}, X_k] = \mathrm{CZ}_{ij}$. Therefore, we can implement a transversal logical $\overline{\mathrm{CZ}}_{ij}$ by applying CZ gates at the level of the physical qubits.

In the context of our stacked 3D surface codes, the above argument implies that we can implement a logical $\overline{\mathrm{CZ}}_{cc'}$ gate by applying CZ gates to the pairs of physical qubits in $\mathcal{SC}_c$ and $\mathcal{SC}_{c'}$ at the vertices of one of the c'' boundaries (our canonical $X_{c''}$ operators are supported on the c'' boundaries).

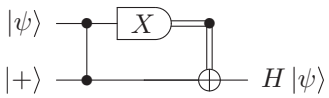


FIG. 17. A circuit which implements a H gate using state preparation, measurement, and CZ [51].

C. Completing a universal set of gates

To achieve universal quantum computing with a $\overline{\text{CCZ}}$ gate we only need a Hadamard gate [$H = (X + Z)/\sqrt{2}$] [50]. The H gate is not transversal in 3D surface codes, but we can still implement it using the teleportation circuit [51] shown in Fig. 17. Therefore, CCZ is universal if we have access to measurement and state preparation in the X and Z bases [52]. As long as we have access to a decoder with a threshold, we can prepare states in the X basis or the Z basis and we can measure qubits in the X basis or the Z basis. We delay discussing decoding strategies for 3D surface codes until Sec. VII. We can generalize the state preparation and measurement methods used in 2D surface codes [3] to 3D surface codes. We quickly review these methods here for completeness. To measure a qubit encoded in a 3D surface code in the Z basis, we simply measure all of the qubits in the code in the Z basis and compute the eigenvalues of all the Z stabilizers. We then correct any X errors implied by this syndrome using a decoder. Finally, we compute the parity of a $\overline{Z}$ operator using the corrected qubit values. To measure in the X basis we replace X with Z (and vice versa) in the procedure we have just described. To fault tolerantly prepare a $|\overline{0}\rangle$ state we prepare each of the physical qubits in the $|0\rangle$ state. We then perform d rounds of error correction (where d is the code distance). To fault tolerantly prepare $|\overline{+}\rangle$ we just replace $|0\rangle$ with $|+\rangle$ in the above procedure.

We can use the circuit in Fig. 17 to implement a single-qubit H gate in a stack of three 3D surface codes and to transfer a logical qubit between different codes in the same stack. We denote the circuit in Fig. 17 as $H_{cc'}$. This circuit takes the state $|\overline{\psi}\rangle_c$ to $\overline{H}\,|\overline{\psi}\rangle_{c'}$. Consider the initial state $|\overline{\psi}\rangle_r\,|\overline{+}\rangle_g\,|\overline{+}\rangle_b$. We can use sequences of $H_{cc'}$ circuits to transfer the state from one code to another or to perform a single-qubit H gate as follows:

$$|\overline{\psi}\rangle_r \xrightarrow{H_{rg}} H\,|\overline{\psi}\rangle_g \xrightarrow{H_{gb}} |\overline{\psi}\rangle_b\,,$$
$$|\overline{\psi}\rangle_r \xrightarrow{H_{rg}} H\,|\overline{\psi}\rangle_g \xrightarrow{H_{gb}} |\overline{\psi}\rangle_b \xrightarrow{H_{br}} H\,|\overline{\psi}\rangle_r\,. \tag{14}$$

VI. 3D SURFACE-CODE LATTICE SURGERY

We have shown how to implement a universal gate set in a single stack of three 3D surface codes. However, in a feasible architecture we also need to be able to transfer qubits between surface codes in different stacks. To accomplish this task we generalize the techniques of 2D surface-code lattice surgery [28,53–55] to 3D surface codes. We note that we will reproduce some material from [28] to make our exposition clearer. Lattice surgery is a code deformation technique which allows us to merge two surface codes into a larger surface code or to split a surface code into two smaller surface codes. Lattice surgery merges and splits can be used for to transfer qubits between codes or to implement CNOT gates. In related recent work, lattice surgery techniques have been extended to the Raussendorf lattice [56], a lattice used in fault-tolerant measurement-based quantum computing [57].

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 [28]). 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.

A. 3D-3D lattice surgery

We start with X-type lattice surgery. Consider two distance d rectified cubic lattices. 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 aligning c boundaries of the two stacks (the rough boundaries of the two codes), preparing a layer of ancillas in the $|0\rangle$ state between the stacks and then measuring new X stabilizers which join the two lattices. The product of these X stabilizers is $\overline{X}_c^{(1)} \otimes \overline{X}_c^{(2)}$ so we learn this value when we perform the merge operation. There may also be new Z stabilizers which we add to the stabilizer group and measure in subsequent rounds. In addition, some Z stabilizers on the boundaries where the merge took place may need to be modified in the new stabilizer group. 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 $\overline{X}_c^{(1)} \otimes \overline{X}_c^{(2)}$ measurement and $|\phi\rangle_c = \alpha\,|0\rangle + \beta\,|1\rangle$ [28]. Any $\overline{X}$ operator for either of the two initial codes is a valid $\overline{X}$ operator for the merged code. However, to form a logical $\overline{Z}$ operator in the new code we must join logical $\overline{Z}$ operators from each of the initial codes into a single string of Z operators which starts and ends at opposite c boundaries. We implement an X-type lattice surgery split by measuring all the qubits in a 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: $\alpha\,|+\rangle_c + \beta\,|-\rangle_c \to \alpha\,|{+}{+}\rangle_c + \beta\,|{-}{-}\rangle_c$ [28]. Figure 18 shows an example of X-type lattice surgery performed on two 3D surface codes.

Z-type lattice surgery is analogous to X-type lattice surgery. To perform a 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 which 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 $\overline{Z}_c^{(1)} \otimes \overline{Z}_c^{(2)}$. The merge implements the mapping $|\psi\rangle_c \otimes |\varphi\rangle_c \to a\,|\psi\rangle_c + (-1)^m bX\,|\psi\rangle_c$, where m is the outcome of the $\overline{Z}_c^{(1)} \otimes \overline{Z}_c^{(2)}$ measurement and $|\varphi\rangle_c = a\,|+\rangle + b\,|-\rangle$ [28]. Any $\overline{Z}$ operator of either original code is a valid $\overline{Z}$ operator of the merged code. However, the valid $\overline{X}$ operators of the merged code are membranes of X operators with boundaries which 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. These measurements implement the following mapping: $\alpha\,|0\rangle_c + \beta\,|1\rangle_c \to \alpha\,|00\rangle_c + \beta\,|11\rangle_c$ [28].

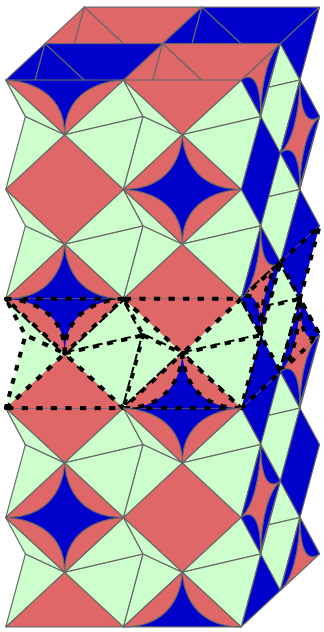


FIG. 18. Lattice surgery in 3D surface codes. Both initial lattices (sublattices with continuous edges) support three surface codes. We prepare a layer of ancilla qubits (vertices of the sublattice with dashed edges) and measure new stabilizers (faces and cells of the sublattice with dashed edges) to merge codes of the same color in separate stacks. To undo a merge, we simply measure the layer of ancilla qubits (vertices of the sublattice with dashed edges). 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 18 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.

We illustrate 3D surface-code lattice surgery with an example. Consider two $d = 3$ rectified cubic lattices placed one above the other as shown in Fig. 18. We add a diamond layer (see Sec. III A 1) of qubits between the two lattices (vertices of the sublattice with dashed edges in Fig. 18). At each vertex we add three qubits (one per code) in the state $|+\rangle_r |0\rangle_g |+\rangle_b$. Next, we merge the stabilizer groups of $\mathcal{SC}_c^{(1)}$ and $\mathcal{SC}_c^{(2)}$, for $c \in \{r, g, b\}$. This implements a Z-type merge on $\mathcal{SC}_r^{(1)}$ and $\mathcal{SC}_r^{(2)}$, an X-type merge on $\mathcal{SC}_g^{(1)}$ and $\mathcal{SC}_g^{(2)}$, and a Z-type merge on $\mathcal{SC}_b^{(1)}$ and $\mathcal{SC}_b^{(2)}$. We now consider each pair of codes with the same color separately and detail how their stabilizer groups transform.

First of all, consider $\mathcal{SC}_g^{(1)}$ and $\mathcal{SC}_g^{(2)}$. The code formed by merging these two codes has nine additional X stabilizers (the complete and incomplete octahedra with dashed edges in Fig. 18). The merged code also has four additional Z stabilizers (the rb faces parallel to the g boundaries in the sublattice with dashed edges in Fig. 18). Some of the Z stabilizers on the boundary are also modified (rb faces in Fig. 18 with dashed and continuous edges). In total, the merged code has 12 additional physical qubits and 13 additional stabilizer generators. The two original codes each had $n = 51$ physical qubits and $n - 1$ stabilizer generators so the merged code has $2n + 12$ physical qubits and $2(n - 1) + 13$ stabilizer generators. Hence, the merged code has a single logical qubit, as required. One can also verify that the product of the new X stabilizers is $\overline{X}_g^{(1)} \otimes \overline{X}_g^{(2)}$.

Next, we consider $\mathcal{SC}_r^{(1)}$ and $\mathcal{SC}_r^{(2)}$. The code formed by merging these two codes has no additional X stabilizers, but some X stabilizers which were present before the merge are modified (r cuboctahedra and rb faces on the b boundaries with dashed and continuous edges in Fig. 18). The merged code has 16 new Z stabilizers associated with the gb faces of the new cuboctahedra (gb faces in the sublattice with dashed edges in Fig. 18). In addition, there are eight new Z stabilizers associated with edges on the r boundaries [blue (dark gray) circular segments with dashed edges in Fig. 18]. However, these new Z stabilizers are not all independent. In the merged lattice, we have four additional complete cuboctahedra and a single additional complete octahedron when compared with the initial lattices. The stabilizers associated with the rg faces of these polyhedra multiply to the identity, so we must remove a stabilizer from the list of new stabilizer generators for each new complete polyhedron. We also have two additional half octahedra whose edges and faces have associated stabilizers which multiply to the identity (see Fig. 9 for an example of such a half octahedron). Therefore, we remove two more stabilizers from the list of new stabilizer generators. Finally, the stabilizers associated with the four edges of rb faces on the r boundaries multiply to the identity, so we must remove half of the new weight-two Z stabilizers from the list of new stabilizer generators. The merged code, therefore, has 13 new stabilizer generators and 12 additional qubits. Hence, the merged code encodes a single logical qubit, as required.

The details the Z-type lattice surgery on $\mathcal{SC}_b^{(1)}$ and $\mathcal{SC}_b^{(2)}$ are the same as the details of Z-type lattice surgery on $\mathcal{SC}_r^{(1)}$ and $\mathcal{SC}_r^{(2)}$ (just exchange r and b in the previous paragraph). To verify that the lattice surgery procedures we have described transform two surface codes into a single surface code for any code distance, all we need to do repeat the analysis of Sec. III A 1 for a slightly different lattice structure. We omit this analysis here as the extension is simple. In Appendix E, we show another possible arrangement of 3D stacks which allows us to do X-type lattice surgery on $\mathcal{SC}_r^{(1)}$ and $\mathcal{SC}_r^{(2)}$, Z-type lattice surgery on $\mathcal{SC}_g^{(1)}$ and $\mathcal{SC}_g^{(2)}$, and Z-type lattice surgery on $\mathcal{SC}_b^{(1)}$ and $\mathcal{SC}_b^{(2)}$.

B. 2D-3D lattice surgery

We can do Z-type lattice surgery on a 2D surface code and a 3D surface code using procedures which are very similar

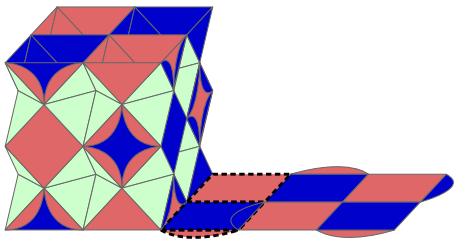


FIG. 19. Z-type lattice surgery on 3D and 2D surface codes (sublattices with continuous edges). We associate X stabilizers with b faces (dark gray) and Z stabilizers with r faces (medium gray) in the 2D surface code. In the 3D stack we consider $\mathcal{SC}_b$ [X stabilizers associated with b cells (dark gray)]. 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 [r faces (medium gray) with dashed edges], whose product is $\overline{Z}_{2D} \otimes \overline{Z}_{3D}$. 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 rb face (medium gray) on the right boundary of the 3D code. This stabilizer is represented by the b face (dark gray) with dashed edges in the figure. To undo the merge operation, we return to measuring the premerge stabilizers.

to 2D surface-code lattice surgery. However, performing X-type lattice surgery on a 2D surface code and a 3D surface code is more complex. This is because the dimension of the $\overline{Z}$ operators in 2D surface codes and 3D surface codes is the same, whereas the dimension of the $\overline{X}$ 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 Fig. 19). To do a lattice surgery merge, we simply measure new Z stabilizers whose product is $\overline{Z}_{2D} \otimes \overline{Z}_{3D}$. The X stabilizers of both codes at the join will also be modified. Figure 19 shows an example Z-type merge of a 3D code and a 2D code. 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 $\overline{Z}$ operators of the original codes are valid $\overline{Z}$ operators of the merged code. However, $\overline{X}$ operators of the merged code are products of membrane operators in the 3D lattice and string operators in the 2D lattice. The merged code is therefore an example of a code with a logical operator which has 2D and 1D parts. We can implement a Z-type split by returning to measuring the premerge stabilizers.

As we previously stated, we can use lattice surgery to implement CNOT gates and to transfer qubits between different surface codes. Consider the initial state $|\psi\rangle|+\rangle$, where $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$. Implementing a Z-type merge between the two qubits followed by a Z-type split produces the state $\alpha|00\rangle + \beta|11\rangle$. If we measure the first qubit in the X basis, $|\psi\rangle$ is transferred to the second qubit (up to a Z correction). The lattice surgery CNOT procedure is similar to the procedure we have just described. Consider the state $|\psi\rangle|+\rangle|\phi\rangle$. To perform a CNOT with $|\psi\rangle$ as the control and $|\phi\rangle$ as the target we first do a Z-type merge of $|\psi\rangle$ and $|+\rangle$ followed by a Z-type split. The second step is to do an X-type merge of $|\phi\rangle$ and $|+\rangle$ followed by an X-type split. There are also some single-qubit corrections that may be necessary which we have omitted. For the full details of this CNOT procedure, see [28]. We can also use a chain of lattice surgery operations to perform a multitarget CNOT gate as shown in [58].

We emphasize that the lattice surgery procedure we have explained in this section is one of many code-deformation procedures which we could use to transfer information from a 3D surface code to a 2D surface code. For example, in Appendix E, we give a different implementation of 2D-3D surface-code lattice surgery. It is also possible to transfer information using a "code-switching" deformation (in the spirit of [59]), where we transform a 3D surface code into a 2D surface code by measuring all but one layer of physical qubits in the X basis. 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.

VII. 3D SURFACE-CODE ARCHITECTURES

In this section, we propose two universal quantum computing architectures which use 3D surface codes. But first we discuss decoding 3D surface codes.

A. Decoding 3D surface codes

Estimating the error thresholds of 3D surface codes is beyond the scope of this article. Instead, we discuss possible decoding strategies for 3D surface codes and reason about the error thresholds we might expect. The 3D surface code is interesting from a decoding point of view because of the asymmetry between membranelike X errors and stringlike Z errors. This asymmetry means that different decoding strategies may be needed for X and Z errors.

We can upper bound the error thresholds of topological codes by relating the codes to condensed-matter models [3]. The phase diagram of the condensed-matter model will then give us an estimate of the optimal error threshold of the code. Using this technique, the optimal error threshold of 2D surface codes has been estimated to be $\approx$11% [60], for a stochastic noise model where X and Z errors happen independently with probability p and measurements are perfect. For the 3D surface code, the optimal error threshold for the same noise model is $\approx$3.3%. We can break this error threshold down further: for a noise model where Z (X) errors happen with probability p and measurements are perfect, the error threshold is $p^Z_{th} \approx 3.3\%$ [61] ($p^X_{th} \approx 23.5\%$ [62,63]).

The above error thresholds will not be achievable in practice due to measurement errors. For the 2D surface code, simulations of the full syndrome extraction circuits indicate an error threshold between 0.5% and 1.1% (see [7] and references therein). To the best of our knowledge, no similar simulation has been performed for 3D surface codes. However, we anticipate that the 3D surface-code error threshold will be lower than the corresponding 2D surface-code error threshold because of the higher dimensionality of the lattice and the larger weight stabilizers in 3D surface codes.

The most popular 2D surface-code decoder is a minimum-weight perfect-matching (MWPM) algorithm [4,64,65]. We could use MWPM to decode Z errors in cubic surface codes and tetrahedral-octahedral surface codes. Alternatively, we

could use the recently proposed Union-Find decoder [66], which has slightly worse performance than MWPM, but a much faster runtime. There are a number of approaches we could take to decoding membranelike X errors in 3D surface codes. Duivenvoorden *et al.* estimated the X-error threshold of 3D cubic surface codes using an efficient renormalization decoder [67]. They found an X-error threshold of $p_{th}^{X} = (17.2 \pm 1)\%$ for an error model with perfect measurements. It would be interesting to generalize the decoder of Duivenvoorden *et al.* to noncubic surface codes such as tetrahedral-octahedral surface codes. Another option for decoding membranelike X errors in 3D surface codes is to use a generalization of Toom's rule as the decoding algorithm [3,68–70]. For such a decoder, Kubica estimated an X-error threshold of $p_{th}^{X} \approx 2\%$ for 3D surface codes with periodic boundaries (3D toric codes) [70]. This error threshold is for a noise model where X errors and measurement errors both occur with probability p. In future work, we intend to extend this result to 3D surface codes with boundaries.

B. Hybrid 2D-3D surface-code architecture

In this section, we present a hybrid 2D-3D surface-code architecture based on [28]. In our hybrid architecture, the main component is a sheet of 2D surface-code patches. Lattice surgery allows us to do CNOT gates between different patches. We can also do Hadamard gates easily as explained in [28]. We use 3D surface codes as CCZ state ($|\mathrm{CCZ}\rangle = \mathrm{CCZ}\,|+++\rangle$) factories in our hybrid architecture, replacing the magic state distillation used in the original architecture. We can fault tolerantly create CCZ states in a 3D surface-code stack as long as we have a decoder with an error threshold. We use Z-type lattice surgery to transfer CCZ states from a stack of 3D surface codes into the sheet of 2D surface codes. There is some subtlety involved in transferring three logical qubits from a single 3D surface-code stack to a 2D surface-code sheet, so we describe this procedure now. We consider a 3D surface-code stack which can interface with a single 2D surface code. This means that we can only transfer encoded states from one of the three 3D surface codes (say $\mathcal{SC}_r$) to the 2D surface code. Other configurations are possible, but we will concentrate on this most basic configuration. We refer to the logical qubit encoded in $\mathcal{SC}_c$ as the c qubit. Assume that we have prepared CCZ state in the 3D surface-code stack. We can transfer the r qubit to the 2D surface code easily using Z-type lattice surgery (see Sec. VI B). Next, we want to transfer the g qubit. We must transfer the state of the g qubit in the stack to the r qubit first. However, we need two qubits in the stack to be ancillas in order to do this [see Eq. (14)], and only one is available. Instead, we transfer the state of the g qubit to the r qubit, with a H gate applied [see Eq. (14)]. Next, we transfer this state to the 2D surface code where we can undo the H gate. Finally, we transfer the state of the b qubit to the r qubit (we now have enough ancillas) and transfer this state to the 2D surface code.

Once we have an encoded CCZ state in our sheet of 2D surface codes, we can implement a CCZ gate on any three qubits using a state injection circuit and some SWAP gates. Figure 20 shows a state injection circuit containing Pauli, H, and CNOT gates that uses one CCZ state to implement a CCZ gate. We constructed this circuit using the methodology described in [51]. To summarize, we have explained how to implement the universal gate set $\{X, Z, H, \mathrm{CNOT}, \mathrm{CCZ}\}$ in our hybrid 2D-3D surface-code architecture.

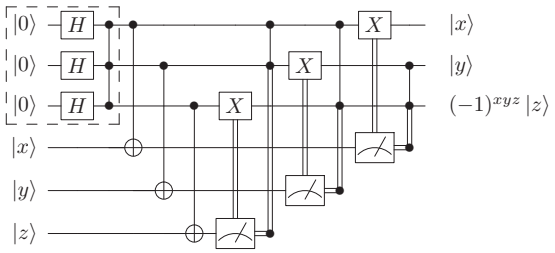


FIG. 20. A circuit that consumes one CCZ state (dashed box) to implement a CCZ gate on the bottom three qubits. We note that $H_t \times \mathrm{CNOT}_{ct} \times H_t = \mathrm{CZ}$, where c and t refer to the control and target qubits.

C. 3D surface-code architecture

In this section, we present a quantum computing architecture where every qubit is encoded in a 3D surface code. We consider a large rectified cubic lattice with 3D "patches," each containing three logical qubits. Each patch is a distance d rectified cubic lattice adjacent to six identical patches. We can do lattice surgery on adjacent patches as described in Sec. VI A. We adopt a Euclidean coordinate system and associate each of the axes with a particular color. For example, we associate the x direction with r which implies that we can do X-type lattice surgery on r qubits (qubits encoded in $\mathcal{SC}_r$ codes) in patches which are adjacent in the x direction. Similarly, we can do X-type lattice surgery on g qubits (b qubits) which are adjacent in the y direction (z direction). This means that we can transfer a qubit from one patch to any of its adjacent patches using X-type or Z-type lattice surgery.

In our architecture we use half of the patches in the lattice as "data patches" and half as "ancilla patches." Data patches contain three logical data qubits and ancilla patches contain three logical ancilla qubits. We can do CNOT gates between any two qubits in data patches which are adjacent to the same ancilla patch using lattice surgery. If the two data qubits have different colors, then we need to use two logical qubits in the ancilla patch during the procedure. For example, imagine we want to do a CNOT between the r qubit (control) and g qubit (target) in the same data patch. First of all, we do a Z-type merge of the r qubit in the data patch and the r qubit in an adjacent ancilla patch. We then undo this merge with a Z-type split. Next, we transfer the state of the r qubit in the ancilla patch to the g qubit in the same ancilla patch, using the procedure in Eq. (14). The next step is to do an X-type merge of the g qubits in the data patch and the ancilla patch. Finally, we undo this merge with an X-type split and apply some Pauli corrections. The procedure we have just described implements a CNOT gate between the r qubit and g qubit in the same data patch.

CNOT gates allow us to swap any two data qubits in data patches which are adjacent to the same ancilla patch. As we have previously shown, we can transversally implement

CZ and CCZ in a single data patch. Finally, we can do a H gate on a single qubit in a data patch by the following method. We first transfer the qubit to an adjacent ancilla patch using lattice surgery. Next, we do a single qubit H using the procedure in Eq. (14) before transferring the qubit back to its original data patch. In the architecture we have just described we can swap arbitrary data qubits and implement a universal gate set in each data patch. 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. This architecture requires no magic state distillation or state injection.

VIII. DISCUSSION

In this article, we introduced the rectified picture of 3D surface codes. We used the rectified picture to analyze stacks of three 3D surface codes, showing that CCZ is transversal in these codes. In addition, we detailed 3D surface-code architectures which allow us to do universal quantum computing without magic state distillation.

As we mentioned in Sec. I, the large resource cost of magic state distillation has motivated research into alternative implementations of non-Clifford gates in topological codes. To reason about the resource scaling of different architectures, we use a space-time overhead metric. Roughly speaking, an architecture which requires n physical qubits and d rounds of syndrome extraction per operation has a space-time overhead of nd. 2D surface-code architectures and 3D gauge color-code architectures have a similar space-time overhead scaling. Distance d 2D surface codes have $O(d^2)$ physical qubits and require $O(d)$ rounds of syndrome extraction to cope with measurement errors. Distance d 3D gauge color codes have $O(d^3)$ physical qubits but only require $O(1)$ rounds of syndrome extraction. The structure of the error syndrome gives us information which we can use to diagnose measurement errors immediately. That is, 3D gauge color codes can be decoded in a single-shot fashion [17,18]. If we want to assess the resource scaling of our 3D surface-code architectures compared with magic state distillation architectures, we need to understand 3D surface-code decoding in more detail. A distance d 3D surface code requires $O(d^3)$ physical qubits but an unknown number of rounds of syndrome extraction. Membranelike X errors in 3D surface codes without boundaries can be decoded using a single-shot cellular automaton decoder [70,70]. However, it seems unlikely that we will be able to use a single-shot decoder to decode stringlike Z errors in 3D surface codes. Nevertheless, due to the links between surface codes and color codes, it may be possible to construct a "3D gauge surface code" where single-shot error correction is possible for both X and Z errors.

It will also be important to estimate the numerical value of the error threshold for both cubic surface codes and tetrahedral-octahedral surface codes. This is because the resources required in a particular architecture depend strongly on the value of the error threshold. The error threshold of the gauge color code has been estimated to be $\approx 0.31\%$ [19], for an error model where qubit errors and measurement errors occur with the same probability. We would expect to observe a smaller error threshold if we were to simulate the full syndrome extraction circuits. Therefore, even with the similar resource scaling, we anticipate that 3D gauge color-code architectures would require more physical qubits than 2D surface-code architectures which use magic state distillation (with current qubit technologies). However, we should note that much more work has gone into optimizing 2D surface-code architectures than gauge color-code architectures, so an error threshold of $p_{th} \approx 0.31\%$ for gauge color codes may be pessimistic. In future work, we plan to investigate decoding 3D surface codes on both cubic and tetrahedral-octahedral lattices. Once we have estimates of the error thresholds we will be able to definitively compare the resources required by 3D surface-code architectures and magic state distillation architectures. It is also interesting to consider an alternative architecture where 3D surface codes and magic state distillation are combined. For example, we could use 3D surface codes to prepare reasonably high-fidelity CCZ states which we would then feed in to a magic state distillation protocol (e.g., [44]). This would remove the need for multiple rounds of magic state distillation and could therefore lead to reduced resource overheads in some scenarios. There are also alternative 3D surface-code architectures we could consider. One of the most popular approaches to 2D surface-code quantum computing is to encode logical qubits as pairs of defects (stabilizers which have been turned off) and braid defects to perform logical gates [4]. It should be possible to generalize this approach to 3D surface codes.

It seems likely that both the rectified picture and code concatenation transformations could be generalized to higher-dimensional ($D \geqslant 4$) surface codes. These generalizations could give us some insight into the structure and transversal gates of higher-dimensional surface codes. Most importantly, the question of whether magic state distillation or transversal gates in 3D topological codes is the best method for promoting 2D topological code architectures to universality remains open. We hope that our work contributes toward answering this question.

ACKNOWLEDGMENTS

The authors would like to thank H. Anwar, E. Campbell, A. Kubica, and P. Webster for helpful discussions. We thank the anonymous referees for helpful comments, and for pointing out a simpler proof of the transversality of CZ in stacked 3D surface codes. M.V. is supported by the Engineering and Physical Sciences Research Council (EPSRC) (Grant No. EP/L015242/1).

APPENDIX A: EXPLICIT CONSTRUCTION OF THE $d = 2$ 3D SURFACE-CODE STACK

Here, we detail list the stabilizer generators and logical operators of three surface codes in the $d = 2$ rectified cubic stack. Figure 21 shows the $d = 2$ rectified cubic lattice. Each code in the stack is a [[12,1,2]] 3D surface code. We label the physical qubits in each code as shown in Fig. 21. P_i denotes a Pauli operator acting on qubit i.

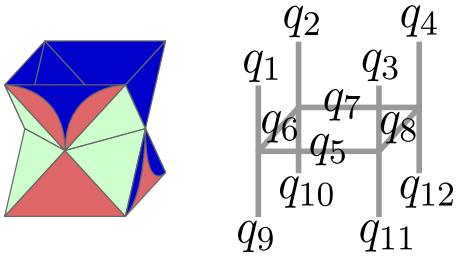


FIG. 21. The $d = 2$ rectified cubic lattice (left) and the $\mathcal{SC}_g$ primal lattice in the Kitaev picture (right), with qubit labels. In the Kitaev picture, qubits are placed on primal lattice edges, X stabilizers are associated with primal lattice vertices, and Z stabilizers are associated with primal lattice faces. $\mathcal{SC}_r$ and $\mathcal{SC}_b$ also have physical qubits at the same locations as the labeled $\mathcal{SC}_g$ physical qubits. Therefore, we use the same label to refer to qubits in different codes that occupy the same position.

The stabilizer generators of $\mathcal{SC}_r$ are

$$\begin{gathered} X_5X_6X_7X_8X_9X_{10}X_{11}X_{12}, \\ X_1X_3X_5,\ X_2X_4X_7, \\ Z_6Z_9,\ Z_6Z_{10}, \\ Z_8Z_{11},\ Z_8Z_{12}, \\ Z_1Z_5Z_6,\ Z_2Z_6Z_7, \\ Z_4Z_7Z_8,\ Z_3Z_5Z_8. \end{gathered} \tag{A1}$$

Example $\mathcal{SC}_r$ logical operators are $\overline{Z}_r = Z_1Z_3$ and $\overline{X}_r = X_3X_4X_8X_{11}X_{12}$.

The stabilizer generators of $\mathcal{SC}_g$ are

$$\begin{gathered} X_1X_5X_6X_9,\ X_2X_6X_7X_{10}, \\ X_3X_5X_8X_{11},\ X_4X_7X_8X_{12}, \\ Z_1Z_3Z_5,\ Z_1Z_2Z_6, \\ Z_2Z_4Z_7,\ Z_3Z_4Z_8, \\ Z_5Z_9Z_{11},\ Z_6Z_9Z_{10}, \\ Z_7Z_{10}Z_{12}. \end{gathered} \tag{A2}$$

Example $\mathcal{SC}_g$ logical operators are $\overline{Z}_g = Z_1Z_9$ and $\overline{X}_g = X_1X_2X_3X_4$.

The stabilizer generators of $\mathcal{SC}_b$ are

$$\begin{gathered} X_1X_2X_3X_4X_5X_6X_7X_8, \\ X_6X_9X_{10},\ X_8X_{11}X_{12}, \\ Z_1Z_5,\ Z_3Z_5, \\ Z_2Z_7,\ Z_4Z_7, \\ Z_5Z_8Z_{11},\ Z_5Z_6Z_9, \\ Z_6Z_7Z_{10},\ Z_7Z_8Z_{12}. \end{gathered} \tag{A3}$$

Example $\mathcal{SC}_b$ logical operators are $\overline{Z}_b = Z_1Z_2$ and $\overline{X}_b = X_1X_3X_5X_9X_{11}$.

APPENDIX B: ALTERNATIVE RECTIFIED CUBIC LATTICE

We can construct an alternative family of stacked 3D surface codes by choosing lattices with different boundaries. The lattices in this family have the global structure of parallelepipeds so we refer to them as parallelepiped lattices. Parallelepiped lattices have two boundaries which slice layers of cuboctahedra in half, like the lattices we discussed in the main text. However, parallelepiped lattices do not have boundaries which slice between layers of cuboctahedra. Instead, they have boundaries which slice one color of cuboctahedra in half and leave the other color of cuboctahedra intact. Figure 22 shows a $d = 3$ parallelepiped lattice. We define three surface codes on parallelepiped lattices by associating X stabilizers with c cells and Z stabilizers with $c'c''$ faces. We must also add some stabilizers on the boundaries to ensure that the boundaries have the correct properties [red (medium gray) and blue (dark gray) circular segments in Fig. 22]. The family of surface codes defined on parallelepiped lattices has the same distance as the family of codes we discussed in the main text. However, the parallelepiped lattices have more physical qubits per logical qubit and are more complex to tessellate.

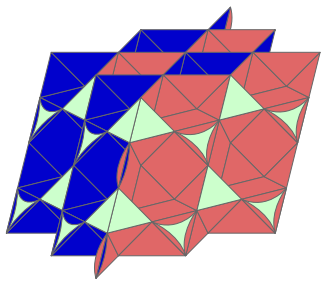


FIG. 22. A lattice from an alternative family of rectified cubic lattices which can support three 3D surface codes. The top and bottom boundaries are the same type as the top and bottom boundaries in Fig. 8. The right boundary slices r cuboctahedra (medium gray) in half and leaves b cuboctahedra (dark gray) intact. The left boundary slices b cuboctahedra (dark gray) in half and leaves r cuboctahedra (medium gray) intact.

APPENDIX C: PROOF OF LEMMA 2

Lemma 2 was required in the proof of the transversality of CCZ for stacked 3D surface codes. We restate it here:

Lemma 2. Given a finite set of k binary vectors $\{a_j\}$ with the same length, the parity of their sum is equal to the sum of their parities.

Proof. An equivalent statement of Lemma 2 is

$$\sum_{j=1}^{k} |a_j| - \left| \sum_{j=1}^{k} a_j \right| = 2t, \tag{C1}$$

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

$$|a_1 + a_2| = |a_1| + |a_2| - 2\mathcal{O}(a_1, a_2), \tag{C2}$$

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 have

$$|a_1| + |a_2| - |a_1 + a_2| = 2\mathcal{O}(a_1, a_2). \tag{C3}$$

Now assume Eq. (C1) is valid for the $k = n$ case. Consider the

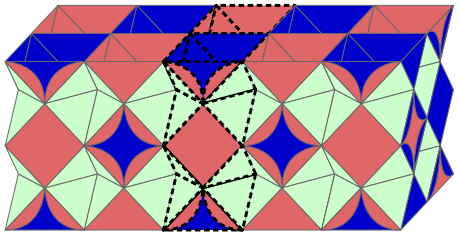


FIG. 23. This configuration of lattices allows us to do X-type lattice surgery on $\mathcal{SC}_r^{(1)}$ and $\mathcal{SC}_r^{(2)}$, Z-type lattice surgery on $\mathcal{SC}_g^{(1)}$ and $\mathcal{SC}_g^{(2)}$, and Z-type lattice surgery on $\mathcal{SC}_b^{(1)}$ and $\mathcal{SC}_b^{(2)}$. Six ancilla qubits are required to do the lattice surgery merges for each pair of codes. The additional (and modified) stabilizers present in the merged codes are associated with the elements of the sublattice with dashed edges.

$k = n + 1$ case:

$$\begin{aligned}\left|\sum_{j=1}^{n+1} a_j\right| &= \left|\sum_{j=1}^{n} a_j\right| + |a_{n+1}| - 2\mathcal{O}\left(\sum_{j=1}^{n} a_j, a_{n+1}\right) \\ &= \sum_{j=1}^{n} |a_j| - 2t + |a_{n+1}| - 2\mathcal{O}\left(\sum_{j=1}^{n} a_j, a_{n+1}\right) \\ &= \sum_{j=1}^{n+1} |a_j| - 2\left[t + \mathcal{O}\left(\sum_{j=1}^{n} a_j, a_{n+1}\right)\right]. \end{aligned} \tag{C4}$$

■

APPENDIX D: TRANSVERSAL CZ IN 2D SURFACE CODES

In this Appendix we show that CZ is transversal for 2D surface codes. Consider a 2D surface code lattice in the rotated picture with faces colored r and b (e.g., Fig. 1). We define a stack of two 2D surface codes on the same lattice. Similarly to the rectified picture of 3D surface codes, we place two physical qubits at each vertex of the lattice (one per code). In the first surface code $\mathcal{SC}_1$, we associate X stabilizers with r faces and Z stabilizers with b faces. In the second surface code $\mathcal{SC}_2$, we associate X stabilizers with b faces and Z stabilizers with r faces. To show that CZ is transversal for this stack of codes, we need to find a transversal operator that implements the following mapping at the logical level:

$$\begin{aligned} I_1 I_2 &\xrightarrow{\text{CZ}} I_1 I_2, \\ Z_j &\xrightarrow{\text{CZ}} Z_j, \\ X_j &\xrightarrow{\text{CZ}} X_j Z_k \quad j \neq k \end{aligned} \tag{D1}$$

where $j, k \in \{1, 2\}$.

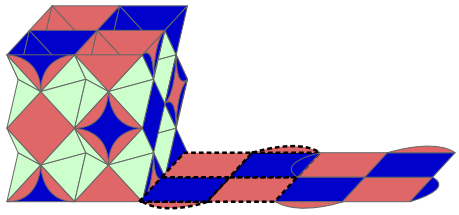


FIG. 24. Z-type lattice surgery on a 3D surface code and a 2D surface code (sublattices with continuous edges). We consider $\mathcal{SC}_b$ in the stack and we associate X stabilizers with b faces (dark gray) and Z stabilizers with r faces (medium gray) in the 2D surface code. Three ancilla qubits are required to do the lattice surgery merge. Four additional Z stabilizers are present in the merged code [r faces (medium gray) with dashed edges] and two X stabilizers from the original codes are modified in the merged code [b faces (dark gray) with dashed edges].

Consider the action of the transversal operator $\overline{\text{CZ}} = \text{CZ}^{\otimes n}$, where n is the number of vertices in the lattice and the CZ gates act on the pairs of qubits at each vertex. Our $\overline{\text{CZ}}$ operator will leave logical $\overline{Z}_j$ operators invariant as these operators consist entirely of Z operators. $\overline{\text{CZ}}$ will map logical $\overline{X}_j$ operators to $\overline{X}_j \overline{Z}_k$ because $\overline{X}_j$ operators consist entirely of X operators and $\overline{X}_j$ operators in one code have the same support as $\overline{Z}_k$ operators in the other code. $\overline{\text{CZ}}$ has no effect on the Z stabilizers of either code but it maps X stabilizers in one code to a tensor product of the original X stabilizer and a Z stabilizer in the other code. To see this, consider an X stabilizer associated with a r face f_r in $\mathcal{SC}_1$. Under the action of $\overline{\text{CZ}}$, this operator is mapped the tensor product of itself and a product of Z operators acting on the $\mathcal{SC}_2$ qubits at the vertices of f_r. This is nothing more than a $\mathcal{SC}_2$ Z stabilizer. Therefore, $\overline{\text{CZ}}$ maps the logical identity to the logical identity. We have shown that $\overline{\text{CZ}} = \text{CZ}^{\otimes n}$ acts as a logical $\overline{\text{CZ}}$, implementing the mapping described in Eq. (D1) at the logical level.

APPENDIX E: ADDITIONAL LATTICE SURGERY EXAMPLES

In this Appendix, we give further examples of 3D surface-code lattice surgery. First, we consider two 3D surface-code stacks, which we denote as $\mathcal{SC}_c^{(i)}$, where $c \in \{r, g, b\}$ denotes the color of the X stabilizers and $i \in \{1, 2\}$ indexes the stack. Figure 23 shows a configuration which allows us to do X-type lattice surgery on $\mathcal{SC}_r^{(1)}$ and $\mathcal{SC}_r^{(2)}$. With this lattice configuration we can also do Z-type lattice surgery on $\mathcal{SC}_b^{(1)}$ and $\mathcal{SC}_b^{(2)}$, and Z-type lattice surgery on $\mathcal{SC}_g^{(1)}$ and $\mathcal{SC}_g^{(2)}$. Figure 24 shows a configuration of lattices which allows us to do lattice surgery on a 3D surface code and a 2D surface code. Unlike the lattice surgery example shown in Fig. 19, this configuration requires ancilla qubits and hence is less efficient.

[1] A. Y. Kitaev, Ann. Phys. (NY) **303**, 2 (2003).
[2] S. B. Bravyi and A. Y. Kitaev, arXiv:quant-ph/9811052.
[3] E. Dennis, A. Kitaev, A. Landahl, and J. Preskill, J. Math. Phys. **43**, 4452 (2002).
[4] A. G. Fowler, M. Mariantoni, J. M. Martinis, and A. N. Cleland, Phys. Rev. A **86**, 032324 (2012).
[5] A. G. Fowler, A. M. Stephens, and P. Groszkowski, Phys. Rev. A **80**, 052312 (2009).

[6] R. Raussendorf and J. Harrington, Phys. Rev. Lett. **98**, 190504 (2007).
[7] A. M. Stephens, Phys. Rev. A **89**, 022321 (2014).
[8] R. Barends, J. Kelly, A. Megrant, A. Veitia, D. Sank, E. Jeffrey, T. C. White, J. Mutus, A. G. Fowler, B. Campbell, Y. Chen, Z. Chen, B. Chiaro, A. Dunsworth, C. Neill, P. O'Malley, P. Roushan, A. Vainsencher, J. Wenner, A. N. Korotkov *et al.*, Nature (London) **508**, 500 (2014).
[9] T. P. Harty, D. T. C. Allcock, C. J. Ballance, L. Guidoni, H. A. Janacek, N. M. Linke, D. N. Stacey, and D. M. Lucas, Phys. Rev. Lett. **113**, 220501 (2014).
[10] S. Vijay, T. H. Hsieh, and L. Fu, Phys. Rev. X **5**, 041038 (2015).
[11] C. D. Hill, E. Peretz, S. J. Hile, M. G. House, M. Fuechsle, S. Rogge, M. Y. Simmons, and L. C. Hollenberg, Sci. Adv. **1**, e1500707 (2015).
[12] J. O'Gorman, N. H. Nickerson, P. Ross, J. J. Morton, and S. C. Benjamin, npj Quantum Inf. **2**, 15019 (2016).
[13] B. Lekitsch, S. Weidt, A. G. Fowler, K. Mølmer, S. J. Devitt, C. Wunderlich, and W. K. Hensinger, Sci. Adv. **3**, e1601540 (2017).
[14] P. W. Shor, SIAM J. Comput. **26**, 1484 (1997).
[15] J. O'Gorman and E. T. Campbell, Phys. Rev. A **95**, 032338 (2017).
[16] S. Bravyi and A. Kitaev, Phys. Rev. A **71**, 022316 (2005).
[17] H. Bombín, New J. Phys. **17**, 083002 (2015).
[18] H. Bombín, Phys. Rev. X **5**, 031043 (2015).
[19] B. J. Brown, N. H. Nickerson, and D. E. Browne, Nat. Commun. **7**, 12302 (2016).
[20] C. Castelnovo and C. Chamon, Phys. Rev. B **78**, 155120 (2008).
[21] A. Kubica, F. Pastawski, and B. Yoshida, New J. Phys. **17**, 083026 (2015).
[22] A. B. Aloshious and P. K. Sarvepalli, Phys. Rev. A **98**, 012302 (2018).
[23] H. Bombín and M. A. Martin-Delgado, Phys. Rev. Lett. **98**, 160502 (2007).
[24] A. Kubica and M. E. Beverland, Phys. Rev. A **91**, 032330 (2015).
[25] S. Bravyi and R. König, Phys. Rev. Lett. **110**, 170503 (2013).
[26] P. Webster and S. D. Bartlett, Phys. Rev. A **97**, 012330 (2018).
[27] B. Criger and B. Terhal, Quantum Inf. Comput. **16**, 1261 (2016).
[28] C. Horsman, A. G. Fowler, S. Devitt, and R. Van Meter, New J. Phys. **14**, 123011 (2012).
[29] S. D. Barrett and P. Kok, Phys. Rev. A **71**, 060310(R) (2005).
[30] K. Fujii, T. Yamamoto, M. Koashi, and N. Imoto, arXiv:1202.6588.
[31] N. H. Nickerson, Y. Li, and S. C. Benjamin, Nat. Commun. **4**, 1756 (2013).
[32] C. Monroe, R. Raussendorf, A. Ruthven, K. R. Brown, P. Maunz, L.-M. Duan, and J. Kim, Phys. Rev. A **89**, 022317 (2014).
[33] N. H. Nickerson, J. F. Fitzsimons, and S. C. Benjamin, Phys. Rev. X **4**, 041041 (2014).
[34] D. Gottesman, Stabilizer codes and quantum error correction, Ph.D. thesis, Caltech, 1997.
[35] X.-G. Wen, Phys. Rev. Lett. **90**, 016803 (2003).
[36] H. Bombín and M. A. Martin-Delgado, Phys. Rev. A **76**, 012305 (2007).
[37] J. T. Anderson, Ann. Phys. (NY) **330**, 1 (2013).
[38] Y. Tomita and K. M. Svore, Phys. Rev. A **90**, 062320 (2014).
[39] H. S. M. Coxeter, *Regular Polytopes*, 3rd ed. (Dover, New York, 1973).
[40] A. Kubica (private communication).
[41] D. Bacon, Phys. Rev. A **73**, 012340 (2006).
[42] D. Poulin, Phys. Rev. Lett. **95**, 230504 (2005).
[43] M. Li, D. Miller, M. Newman, Y. Wu, and K. R. Brown, Phys. Rev. X **9**, 021041 (2019).
[44] A. Paetznick and B. W. Reichardt, Phys. Rev. Lett. **111**, 090505 (2013).
[45] H. Bombín and M. A. Martin-Delgado, Phys. Rev. Lett. **97**, 180501 (2006).
[46] H. Bombín, G. Duclos-Cianci, and D. Poulin, New J. Phys. **14**, 073048 (2012).
[47] N. Delfosse, Phys. Rev. A **89**, 012317 (2014).
[48] H. Bombín, Commun. Math. Phys. **327**, 387 (2014).
[49] A. B. Aloshious, A. N. Bhagoji, and P. K. Sarvepalli, arXiv:1804.00866.
[50] Y. Shi, Quantum Inf. Comput. **3**, 84 (2003).
[51] X. Zhou, D. W. Leung, and I. L. Chuang, Phys. Rev. A **62**, 052316 (2000).
[52] T. J. Yoder, R. Takagi, and I. L. Chuang, Phys. Rev. X **6**, 031039 (2016).
[53] D. Litinski and F. v. Oppen, Quantum **2**, 62 (2018).
[54] A. G. Fowler and C. Gidney, arXiv:1808.06709.
[55] D. Litinski, Quantum **3**, 128 (2019).
[56] D. Herr, A. Paler, S. J. Devitt, and F. Nori, Quantum Sci. Technol. **3**, 035011 (2018).
[57] R. Raussendorf, J. Harrington, and K. Goyal, Ann. Phys. (NY) **321**, 2242 (2006).
[58] D. Herr, F. Nori, and S. J. Devitt, New J. Phys. **19**, 013034 (2017).
[59] H. Bombín, New J. Phys. **18**, 043038 (2016).
[60] A. Honecker, M. Picco, and P. Pujol, Phys. Rev. Lett. **87**, 047201 (2001).
[61] T. Ohno, G. Arakawa, I. Ichinose, and T. Matsui, Nucl. Phys. B **697**, 462 (2004).
[62] Y. Ozeki and N. Ito, J. Phys. A: Math. Gen. **31**, 5451 (1998).
[63] M. Hasenbusch, F. Parisen Toldin, A. Pelissetto, and E. Vicari, Phys. Rev. B **76**, 184202 (2007).
[64] J. Edmonds, Can. J. Math. **17**, 449 (1965).
[65] V. Kolmogorov, Math. Program. Compuation **1**, 43 (2009).
[66] N. Delfosse and N. H. Nickerson, arXiv:1709.06218.
[67] K. Duivenvoorden, N. P. Breuckmann, and B. M. Terhal, IEEE Trans Inf. Theory **65**, 2545 (2018).
[68] A. L. Toom, Adv. Prob. **6**, 549 (1980).
[69] A. Kubica and J. Preskill, arXiv:1809.10145.
[70] 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.