**THE SURFACE CODE**

Ben Reichardt

**Intuition:** (scale-invariant superpositions of) String nets

1. Start with a surface (2-manifold with boundary)

2. Draw a net on it (cycles or degree 4)
2. Codeword = uniform superposition overall such pictures

Discretization

- pbit for every lattice edge
- $|1\rangle = \text{net edge}$
- $|0\rangle = \text{none}$

This gives a quantum code!

A. Codewords
B. Protects against
C. Stabilizers
This gives a quantum code:

A. Codewords

B. Protects against errors

C. Stabilizers

\[ |\text{encoded}\rangle_{000} = \begin{array}{c} \begin{array}{c} \text{circles} \\ \text{hatched} \end{array} \end{array} + \begin{array}{c} \begin{array}{c} \text{circles} \\ \text{red} \end{array} \end{array} + \ldots \]

\[ |\text{encoded}\rangle_{100} = \begin{array}{c} \begin{array}{c} \text{circles} \\ \text{red} \end{array} \end{array} + \begin{array}{c} \begin{array}{c} \text{circles} \\ \text{hatched} \end{array} \end{array} + \ldots \]

\[ |\text{encoded}\rangle_{110} = \begin{array}{c} \begin{array}{c} \text{circles} \\ \text{red} \end{array} + \begin{array}{c} \begin{array}{c} \text{circles} \\ \text{hatched} \end{array} + \ldots \]

Protects against errors:

Any region just looks like

\[ \therefore \text{you can't tell if there are an even (10\rangle) or odd (11\rangle) number of loops around a hole} \]

Logical operators

logical X on first encoded qubit

(meaning $X \otimes I$ elsewhere)

switches first qubit

$10 \leftrightarrow 11$
distance (minimum weight of an operator acting nontrivially on the codespace) = \min \{ \text{circumference of a hole}, \text{distance between holes, or from hole to boundary} \} \\

Stabilizers (parity checks satisfied by codewords)

Rule 1: Even net degree at every vertex

Rule 2: All cycles have equal amplitude

To force \( \alpha = \beta \), use the stabilizer \\
\[
\begin{array}{c|c}
X \\text{this either creates} \\
\hline
\end{array}
\]

a cycle with the same
A big cycle is created by multiplying the stabilizers for the tiles it encloses.

Observe: The stabilizers are local!
(codespace = ground space of local Hamiltonian)
--- Important for physical implementation

- code qubits
- qubits used to measure
- vertex stabilizers
- tile stabilizers

(Observe: Mathematically, the logical operators commute with the stabilizers; this is why they leave the code unchanged.)
Observer: Mathematically, the logical operators commute with the stabilizers; this is why they leave the code unchanged.
\[ P \otimes X \rangle = X (P \otimes X \rangle) = X 1 \rangle \]

Codespace = ground-space of Hamiltonian
\[ \mathcal{H} = - \sum_{\text{vertices}} \sum_{\text{tiles}} \]
- all terms commute
- 4-local, and geometrically local in 2D

**How to use this code?**
1. How to correct errors
2. How to correct errors fault tolerantly?
3. How to compute on the encoded data fault tolerantly?

**Errors and error correction**

X error

2 X errors

X errors create strings — undetectable in the interior, but detectable at the endpoints

Z errors can’t be drawn as nets, but are completely symmetrical: chains of Z errors (on the dual lattice) show up at their endpoints
Error correction uses minimum-weight matching. For every vertex, consider its parity (should be even) many explanations:

- Observed odd-parity vertices
- Weight-13 error
- Best explanation (not unique):
  - Weight-5 error

Can correct X and Z errors separately. (I am ignoring some subtleties)

**Fault-tolerant error correction**

How do we explain?

- Observed odd-parity vertices

⇒ There must have been an error measuring the parity of a vertex

**Solution:** Repeat "syndrome extraction," and run matching algorithm also in time!

Error chains can grow in space and time, with syndrome flips observed at the endpoints.

**Fault-tolerant computation**
Most interesting gate: CNOT (entangling)

\[ \begin{array}{c}
\text{a} \\
\text{b}
\end{array} \quad \begin{array}{c}
\text{a} \\
\text{a} \oplus \text{b}
\end{array} \]

**Method 1: Transversal gates**

Observe: Sum mod 2 of two valid net diagrams is another valid diagram.

\[ \begin{array}{c}
\text{T} \\
\oplus
\end{array} \quad \begin{array}{c}
\text{T}
\end{array} = \begin{array}{c}
\text{T}
\end{array} \]

**Corollary:** Transversal CNOTs implement encoded CNOT.

(a loop around the top hole will be copied around the bottom hole)

**Method 2: Code deformation**

**Idea:**

Code \[\rightsquigarrow\] slightly different code (different surface) \[\rightsquigarrow\] Original code — but with manipulated codespace

2013-2014 USC notes Page 8
Smooth and rough boundary conditions

- rough boundary
  = dual of smooth bdry
  - allows net lines to terminate
First bit (0 or 1) is XORed into 2nd bit

CNOT gate

Typical braiding for CNOT gate, in time

2D Architecture for a quantum computer

- Smooth qubit being expanded to braid around rough qubit
- Lower-distance qubits involved in 1st level of distillation (with higher logical error rate)
Elaborations:

- Surface code on other lattices
- Complexity of min-wt matching: $O(n^2)$ by Edmond
- (non-fault tolerant) Syndrome extraction
- Codewords $107$ and $117$ in the computational basis
- Obtaining a universal gate set