Shapes, Software, Symmetries

Queens' Maths Society
13th November, 2019
Nicolaus Heuer

Little Bio

- 2010-2015: BSc, MSc, ETH Zurich
- 2015-2019: Dphil (PhD), Oxford
- 2019-now: Herchel Smith Fellow and PDRA at Queens’, Cambridge.

Motivating Question

Motivating Question

Can I tell two shapes apart?

Motivating Question

Can I tell two shapes apart?

Answer: No! If the shapes have dimensions 4 or more.

Outline:

- Geometric Group Theory

Outline:

- Geometric Group Theory
- Explain the Title:
  - Shapes
  - Software
  - Symmetries

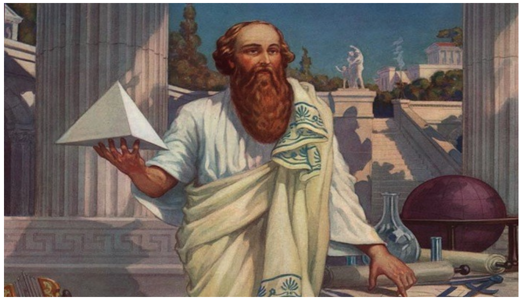
Outline:

- Geometric Group Theory
- Explain the Title:
  - Shapes: Manifolds
  - Software: Turing Machines
  - Symmetries: Groups

Outline:

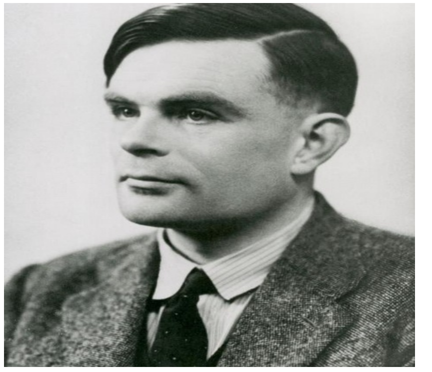
- Geometric Group Theory
- Explain the Title:
  - Shapes: Manifolds
  - Software: Turing Machines
  - Symmetries: Groups
- From Software to Symmetries
- From Symmetries to Shapes

Geometry

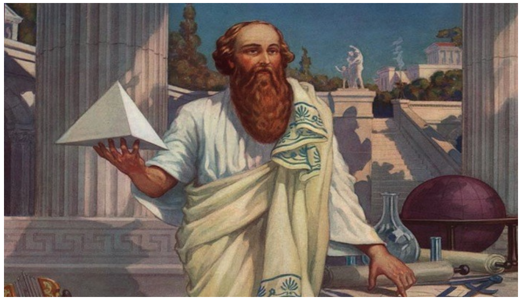


Group Theory

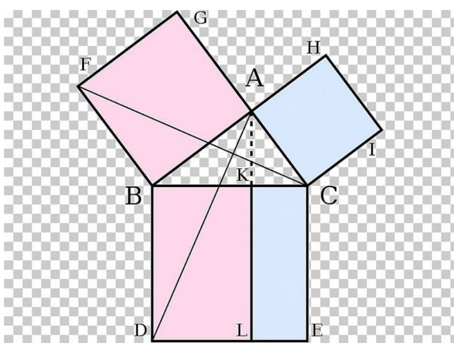
Computer Science



Geometry



Pythagoras of Samos
(c. 570 – c. 495 BC)



Group Theory

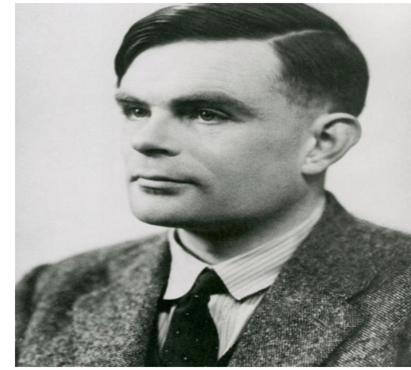
Évariste Galois
(1811 – 1832)

$$ax^2 + bx + c = 0$$

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

$$x^5 - 4x + 2 = 0$$

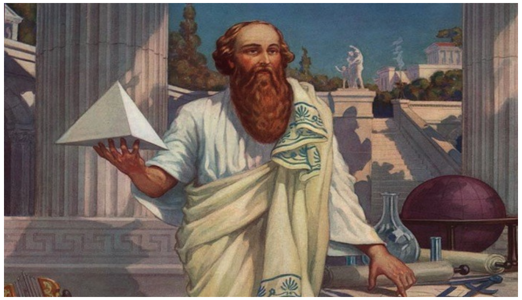
Computer Science



Alan Turing
(1912-1957)

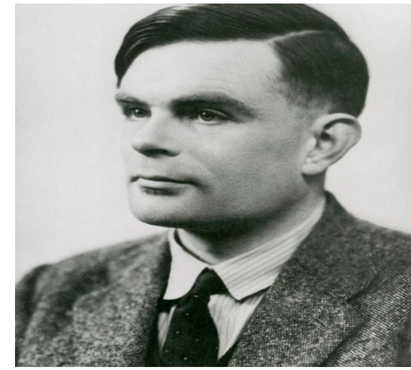
$$h(i,x) = \begin{cases} 1 & \text{if program i halts on input x} \\ 0 & \text{else.} \end{cases}$$

Geometry

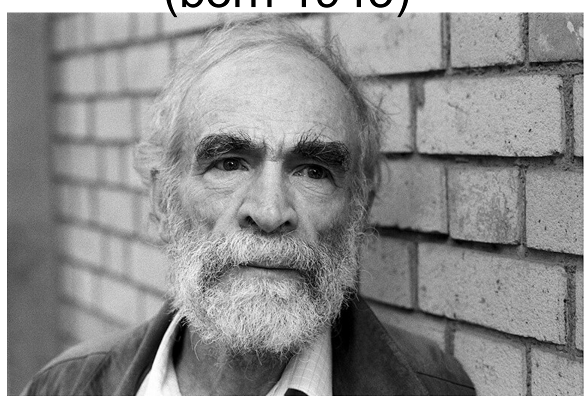


Group Theory

Computer Science



Mikhail Gromov
(born 1943)



Geometric Group Theory

Geometry	Group Theory	Computer Science

- curvature
- growth
- length
- volume

↔

- Abelian
- Nilpotent
- Cyclic subgrouops
- Hyperbolic groups

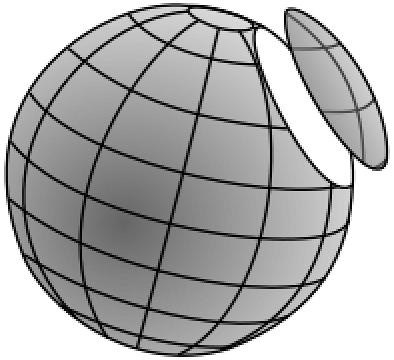
↔

- Decision Problems
- Computational Complexity

Geometric Group Theory

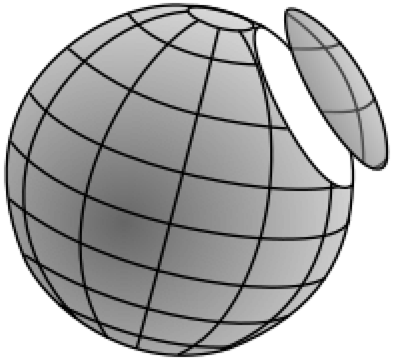
Shapes
(Manifolds)

A **surface** is an object which locally looks like a disk:

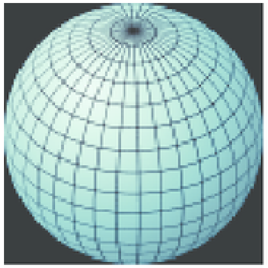
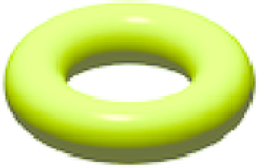
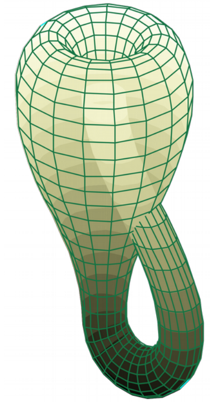


Shapes
(Manifolds)

A **surface** is an object which locally looks like a disk:

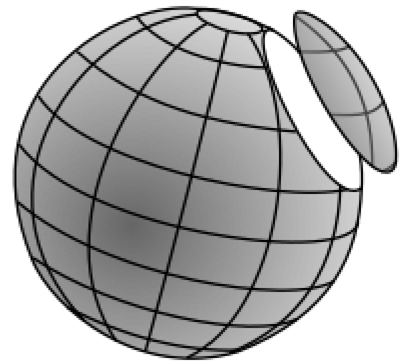


It is possible to *list* all surfaces:

     ...

Shapes
(Manifolds)

A **surface** is an object which locally looks like a disk:



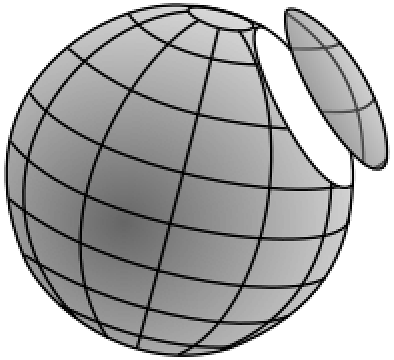
It is possible to *list* all surfaces:

Theorem: If X is a closed (compact+no-boundary) surface then X is equivalent to one of the following families of surfaces

- The sphere
- S(g), the surface with g handles
- P(g), a Klein bottle with g handles.

Shapes
(Manifolds)

A **surface** is an object which locally looks like a disk:



???

It is possible to *list* all surfaces:

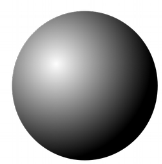
Theorem: If X is a closed (compact+no-boundary) surface then X is equivalent to one of the following families of surfaces

- The sphere
- S(g), the surface with g handles
- P(g), a Klein bottle with g handles.

Shapes
(Manifolds)

Equivalent surfaces:

=



=

=

Shapes
(Manifolds)

For which n can I make a list (without repetitions) of all n-manifolds?

Shapes
(Manifolds)

Examples:
- 2-dimensional manifolds: possible to classify
- 3-dimensional manifolds: possible to classify
- 4-dimensional manifolds: impossible to classify
- 5-dimensional manifolds: impossible to classify
- …

Uses lots of group theory!!

Shapes
(Manifolds)

Examples:
- 2-dimensional manifolds: possible to classify
- 3-dimensional manifolds: possible to classify
- 4-dimensional manifolds: impossible to classify
- 5-dimensional manifolds: impossible to classify
- …

Uses lots of group theory!!

We will explain this today!

Software
(Turing Machines)

What does a computer do?

Software
(Turing Machines)

What does a computer do?

Input →  → Output

Software
(Turing Machines)

What does a computer do?

Input →  → Output

Computation:
- Has **infinite** memory
- **Finite** set of rules to manipulate entries in memory

Software
(Turing Machines)

Formally:

- Memory: Infinite **tape** with a start point and a finite set with symbols from a finite set A (e.g. A = {0,1,b}) on them. A is the **tape alphabet**.
- Working-Memory: A finite set Q of **states**. With one initial state i and one final state f.
- A **pointer** to the tape
- A set of **rules** $r : Q \times A \longrightarrow Q \times A \times \{L,R\}$

How to compute things:

- Write input on the tape, set pointer to the start and state to i
- Manipulate entries according to rules.
- Finish: once the state reaches f. Then the output is on the tape.

Software
(Turing Machines)

Formally:

- Memory: Infinite **tape** with a start point and a finite set with symbols from a finite set A (e.g. A = {0,1,b}) on them. A is the **tape alphabet**.
- Working-Memory: A finite set Q of **states**. With one initial state i and one final state f.
- A **pointer** to the tape
- A set of **rules** $r : Q \times A \longrightarrow Q \times A \times \{L,R\}$

Input: 0 1 1 0 0 0 1 0 1 1 0

Software
(Turing Machines)

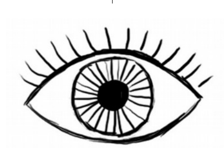
Formally:

- Memory: Infinite **tape** with a start point and a finite set with symbols from a finite set A (e.g. A = {0,1,b}) on them. A is the **tape alphabet**.
- Working-Memory: A finite set Q of **states**. With one initial state i and one final state f.
- A **pointer** to the tape
- A set of **rules** $r : Q \times A \longrightarrow Q \times A \times \{L,R\}$

Input: 0 1 1 0 0 0 1 0 1 1 0

b	b	0	1	1	0	0	0	1	0	1	1	0	b	b

State = i



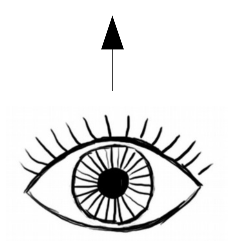
r(i,0) = (q_1,1,L)

Software
(Turing Machines)

Formally:

- Memory: Infinite **tape** with a start point and a finite set with symbols from a finite set A (e.g. A = {0,1,b}) on them. A is the **tape alphabet**.
- Working-Memory: A finite set Q of **states**. With one initial state i and one final state f.
- A **pointer** to the tape
- A set of **rules** r : Q x A $\rightarrow$ Q x A x {L,R}

b	b	0	1	1	0	1	0	1	0	1	1	0	b	b



State = q_1

r(i,0) = (q_1,1,L)

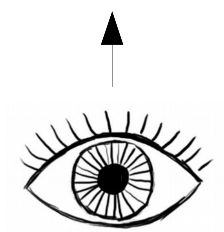
Software
(Turing Machines)

Formally:

- Memory: Infinite **tape** with a start point and a finite set with symbols from a finite set A (e.g. A = {0,1,b}) on them. A is the **tape alphabet**.
- Working-Memory: A finite set Q of **states**. With one initial state i and one final state f.
- A **pointer** to the tape
- A set of **rules** r : Q x A $\longrightarrow$ Q x A x {L,R}

b	b	0	1	1	0	1	0	1	0	1	1	0	b	b

State = q_1



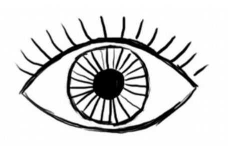
r(q_1,0) = (q_2,0,L)

Software
(Turing Machines)

Formally:

- Memory: Infinite **tape** with a start point and a finite set with symbols from a finite set A (e.g. A = {0,1,b}) on them. A is the **tape alphabet**.
- Working-Memory: A finite set Q of **states**. With one initial state i and one final state f.
- A **pointer** to the tape
- A set of **rules** r : Q x A $\longrightarrow$ Q x A x {L,R}

b	b	0	1	1	0	1	0	1	0	1	1	0	b	b



State = q_2

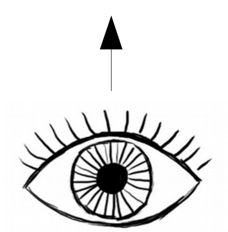
r(q_1,0) = (q_2,0,L)

Software
(Turing Machines)

Formally:

- Memory: Infinite **tape** with a start point and a finite set with symbols from a finite set A (e.g. A = {0,1,b}) on them. A is the **tape alphabet**.
- Working-Memory: A finite set Q of **states**. With one initial state i and one final state f.
- A **pointer** to the tape
- A set of **rules** r : $Q \times A \rightarrow Q \times A \times \{L,R\}$

b	b	0	1	1	0	1	0	1	0	1	1	0	b	b



State = q_2

r(q_2,1) = (f,1,R)

Software
(Turing Machines)

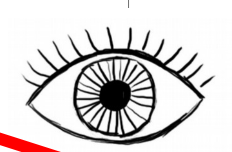
Formally:

- Memory: Infinite **tape** with a start point and a finite set with symbols from a finite set A (e.g. A = {0,1,b}) on them. A is the **tape alphabet**.
- Working-Memory: A finite set Q of **states**. With one initial state i and one final state f.
- A **pointer** to the tape
- A set of **rules** r : Q x A $\longrightarrow$ Q x A x {L,R}

b	b	0	1	1	0	1	0	1	0	1	1	0	b	b

r(q_2,1) = (f,1,R)

State = f



COMPUTATION FINISHED!

Software
(Turing Machines)

Does a given Turing machine has to complete its computation (‘Halt’)?
→ Clearly No!

Can we decide if a Turing machine halts?

Is there a Turing machine h with input all Turing Machines and all inputs, such that:

$$h(i,x) = \begin{cases} 1 & \text{if program i halts on input x} \\ 0 & \text{else.} \end{cases}$$

Software
(Turing Machines)

Does a given Turing machine has to complete its computation (‘Halt’)?
→ Clearly No!

Can we decide if a Turing machine halts?

Is there a Turing machine h with input all Turing Machines and all inputs, such that:

h : N x N → {0,1}

Assume that all Turing Machines and Inputs are encoded in N (natural numbers).

$$h(i,x) = \begin{cases} 1 & \text{if program i halts on input x} \\ 0 & \text{else.} \end{cases}$$

Software
(Turing Machines)

$$h(i,x) = \begin{cases} 1 & \text{if program i halts on input x} \\ 0 & \text{else.} \end{cases}$$

Define: g : N → {0,1} via:

$$g(i) = \begin{cases} 0 & \text{if } h(i,i) = 0 \\ \texttt{loop forever} & \text{else.} \end{cases}$$

If g is the nth Turing machine then

Software
(Turing Machines)

$$h(i,x) = \begin{cases} 1 & \text{if program i halts on input x} \\ 0 & \text{else.} \end{cases}$$

Define: g : N → {0,1} via:

$$g(i) = \begin{cases} 0 & \text{if } h(i,i) = 0 \\ \texttt{loop forever} & \text{else.} \end{cases}$$

If g is the n^{th} Turing machine then

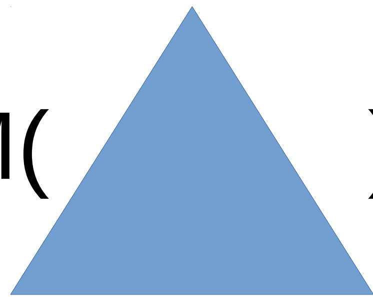
- If g(n) halts, then h(n,n)=0, hence g(n) does not halt
- If g(n) does not halt, then h(n,n)=1, hence g(n) does halt.

CONTRADICTION

Software
(Turing Machines)

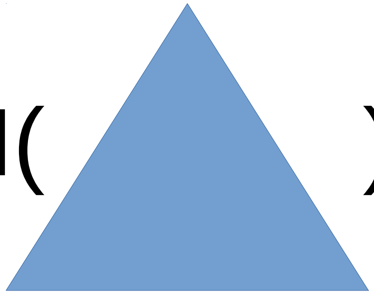
Theorem (Turing): There is no Turing Machine which decides if a Turing Machine with a given input halts or not.

Symmetries
(Groups)

ISOM() = {e, r1, r2, s1, s2, s3}

= D6

Symmetries
(Groups)

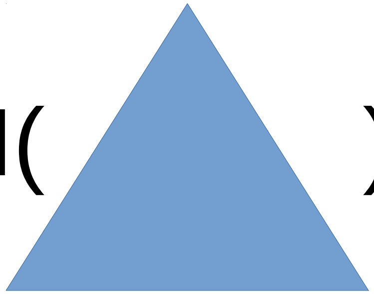
ISOM() = {e, r1, r2, s1, s2, s3}

= D6

How can we store this information?

- Write out the Cayley Table (not so efficient)

Symmetries
(Groups)

ISOM() = {e, r1, r2, s1, s2, s3}

= D6

How can we store this information?

- Write out the Cayley Table (not so efficient)
- Or: Just record the most essential equations (‘*relations*’):

$$D_6 \cong \langle r, s \mid r^3 = 1, s^2 = 1, srs^{-1} = r^{-1} \rangle$$

Symmetries
(Groups)

Generally: S a finite set, R a set of relations in S, then set:

$$G \cong \langle S \mid R \rangle := F(S)/\langle\langle R \rangle\rangle$$

G is *finitely presented* if S and R are finite.

Symmetries
(Groups)

Generally: S a finite set, R a set of relations in S, then set:

$$G \cong \langle S \mid R \rangle := F(S)/\langle\langle R \rangle\rangle$$

G is *finitely presented* if S and R are finite.

What can we know about the group from a given group presentation?

From Software to Symmetries

Question: Can we decide if an element in a finitely presented group is trivial?

From Software to Symmetries

Question: Can we decide if an element in a finitely presented group is trivial?

NO!

Because we can use groups to simulate Turing Machines

From Software to Symmetries

(sketch in semi-groups)

Let T be a Turing machine with

- Tape Alphabet A
- States Q

From Software to Symmetries

(sketch in semi-groups)

Let T be a Turing machine with

- Tape Alphabet A
- States Q

Define a finitely preseted (semi) group via

$$G \cong \langle A \cup Q \mid R_T \rangle$$

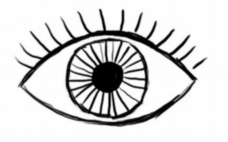
From Software to Symmetries

(sketch in semi-groups)

Input: 0 1 1 0 0 0 1 0 1 1 0

b	b	0	1	1	0	0	0	1	0	1	1	0	b	b

State = i



r(i,0) = (q_1,1,L)

Word in G:

0 1 1 0 P i 0 1 0 1 1 0

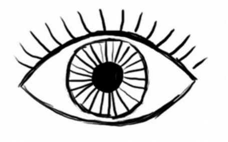
From Software to Symmetries

(sketch in semi-groups)

Input: 0 1 1 0 0 0 1 0 1 1 0

b	b	0	1	1	0	0	0	1	0	1	1	0	b	b

State = i



r(i,0) = (q_1,1,L)

Relation:
0 P i 0 = P q_1 0 1

Word in G:

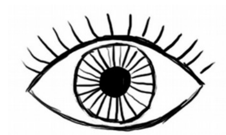
0 1 1 0 P i 0 1 0 1 1 0

From Software to Symmetries

(sketch in semi-groups)

b	b	0	1	1	0	1	0	1	0	1	1	0	b	b

State = q_1



r(q_1,0) = (q_1,1,L)

Relation:

0 P i 0 = P q_1 0 1

Word in G:

0 1 1 0 P i 0 1 0 1 1 0

= 0 1 1 P q_1 0 1 1 0 1 1 0

From Software to Symmetries

Theorem: It is undecidable if an element in a finitely presented group represents the identity or not.

From Symmetries to Shapes

From Symmetries to Shapes

(and the other way around!)

From Symmetries to Shapes

Fundamental group:

Let X be a shape (a topological space) with a point x.

p(X) the *fundamental group of X is:*

- Set of all loops based at x
- We can combine two loops g, h in p(X) by applying them after each other

From Symmetries to Shapes

Fundamental group:

Let X be a shape (a topological space) with a point x.

p(X) the *fundamental group of X is:*

- Set of all loops based at X
- We can combine two loops g, h in p(X) by applying them after each other
- Equivalent if loops can be stretched into each other.

Group?

- Identity?
- Associativity?
- Inverse?

From Symmetries to Shapes

Fundamental group:

Let X be a shape (a topological space) with a point x.

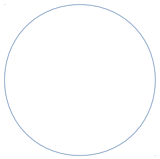
p(X) the *fundamental group of X is:*

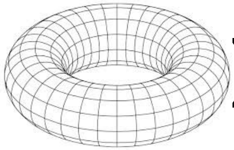
- Set of all loops based at X
- We can combine two loops g, h in p(X) by applying them after each other
- Equivalent if loops can be stretched into each other

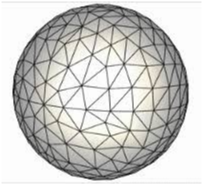
Group?

- Identity? Loop which remains at x
- Associativity? Clear!
- Inverse? Same loop in opposite orientation

From Symmetries to Shapes

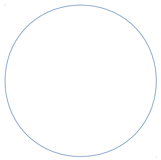
p() =

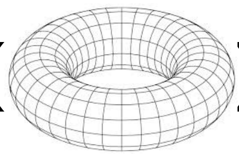
p() =

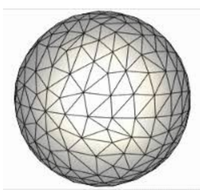
p() =

p(M), M manifold =

From Symmetries to Shapes

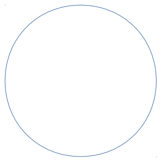
p() = Z $\cong \langle a \mid \rangle$

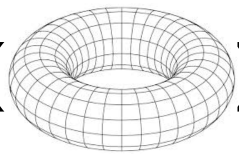
p() = Z x Z $\cong \langle a, b \mid ab = ba \rangle$

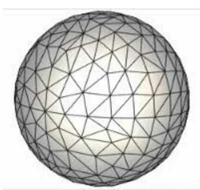
p() = {e}

p(M), M manifold = finitely presented group

From Symmetries to Shapes

p() = Z $\cong \langle a \mid \rangle$

p() = Z x Z $\cong \langle a, b \mid ab = ba \rangle$

p() = {e}

p(M), M manifold = finitely presented group

Manifolds ⟶ Groups

From Symmetries to Shapes

Theorem: Let G be a finitely presented group and let n > 3. Then there is a compact n-manifold M, which can be effectively computed, such that p(M) = G.

Putting Things Together

- If I could decide invariants in 4-manifolds I can decide invariants in finitely presented groups
- If I can decide invariants in finitely presented groups I can decide halting for turing machines.

Question?