A new weakly universal cellular automaton in the $3D$ hyperbolic space with two states

Maurice Margenstern

UPVM, Metz,
LITA, EA 3097
LORIA, UMR 7503, CNRS

RP'2011

September, 28-30, 2011

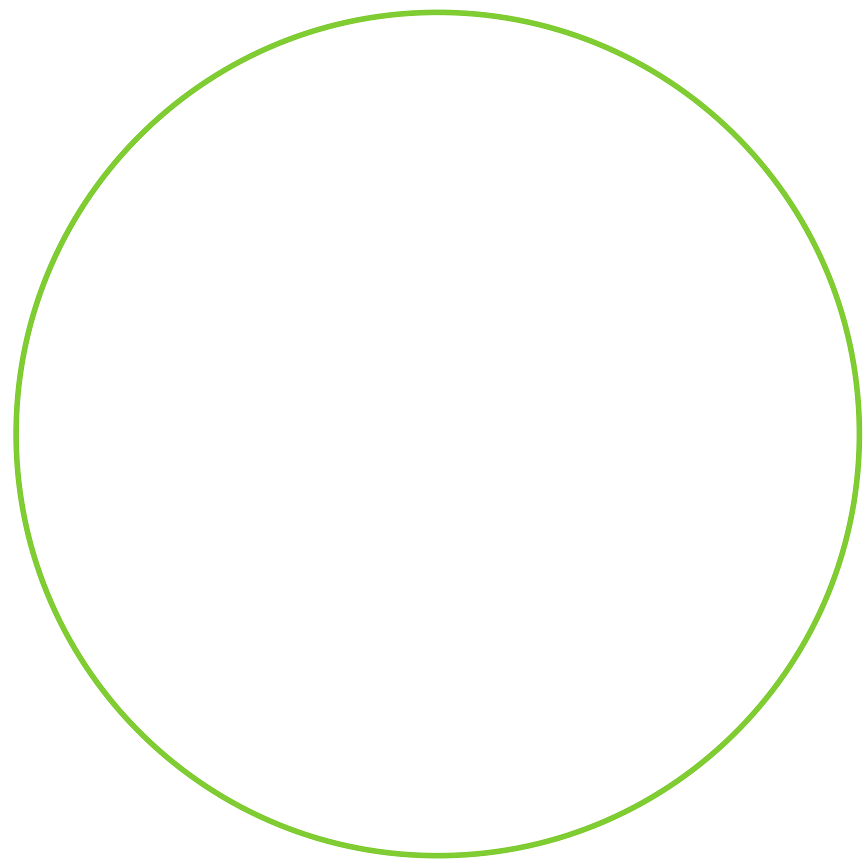
Genova, ITALY

in this talk:

1. recall from hyperbolic geometry
2. pentagrid and dodecagrid
3. CA's in the dodecagrid
4. railway simulation
5. strong/weak universality in CA's
6. a universal CA in the dodecagrid
   with 2 states

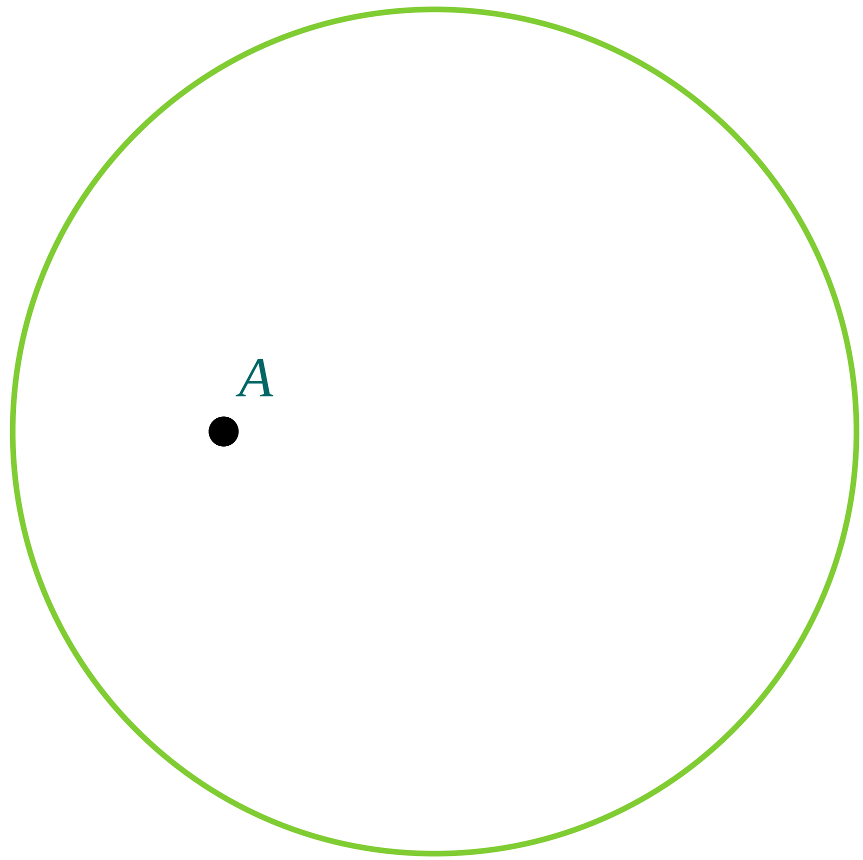
1. recall from hyperbolic geometry

Poincaré's disc model



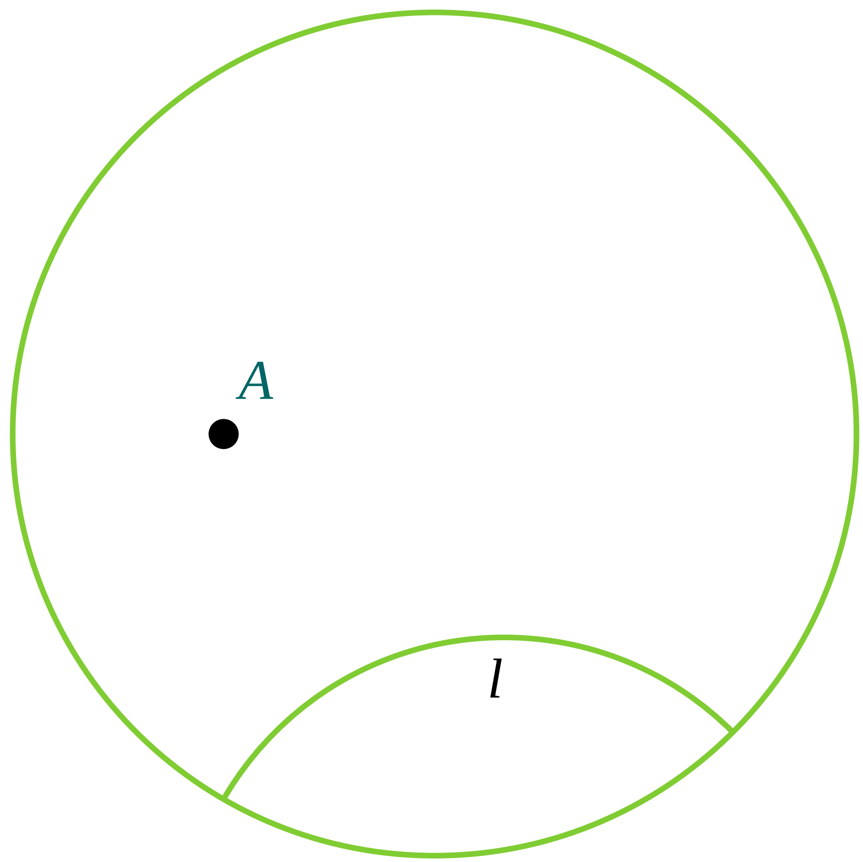
Poincaré's disc model

a point A



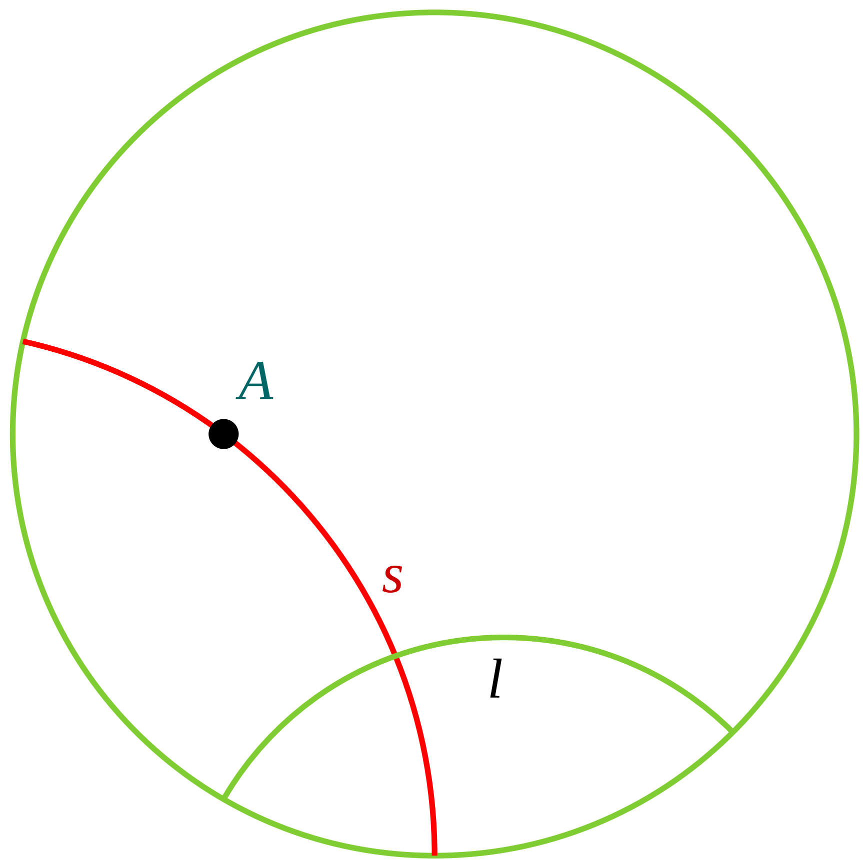
Poincaré's disc model

a point A
a line ℓ



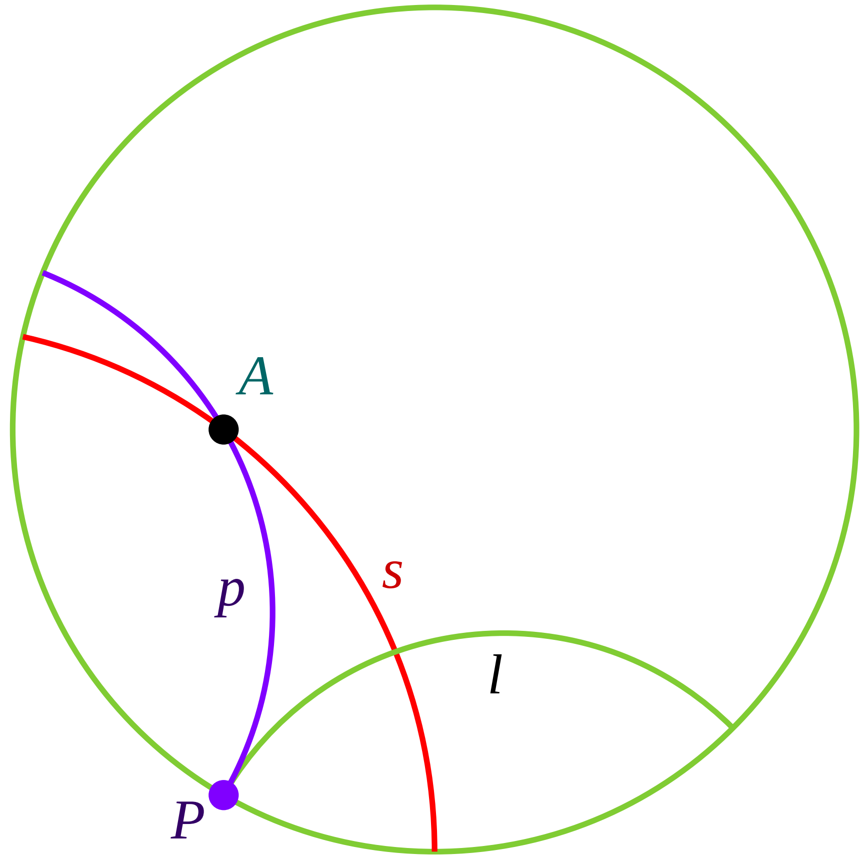
Poincaré's disc model

a **secant**
through A
which cuts ℓ



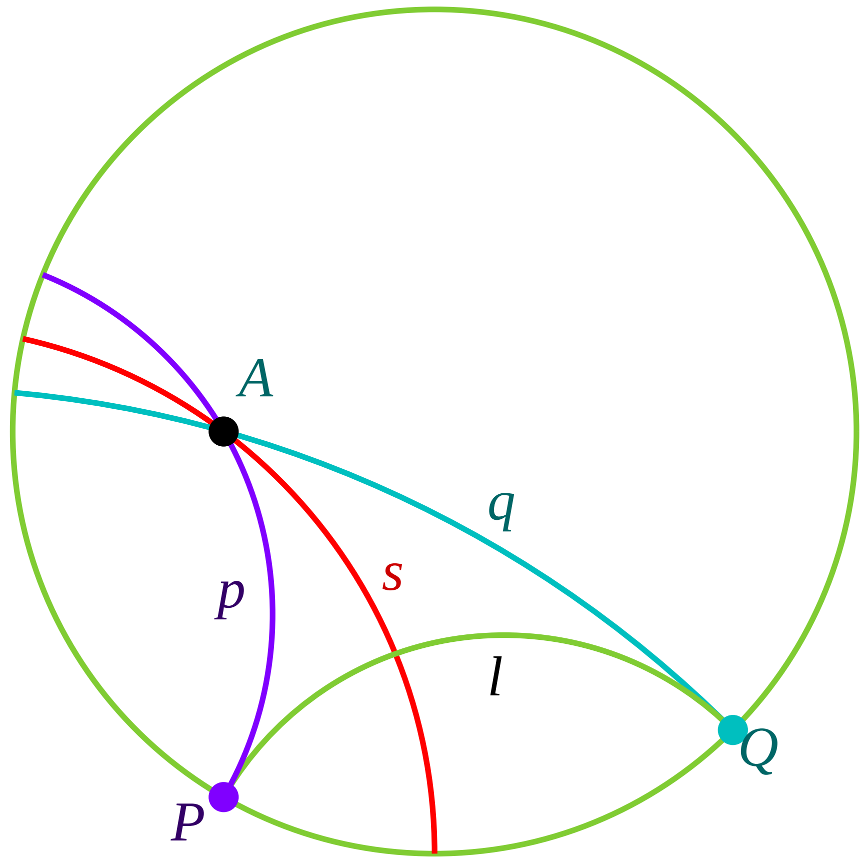
Poincaré's disc model

a **parallel** p
to ℓ
through A



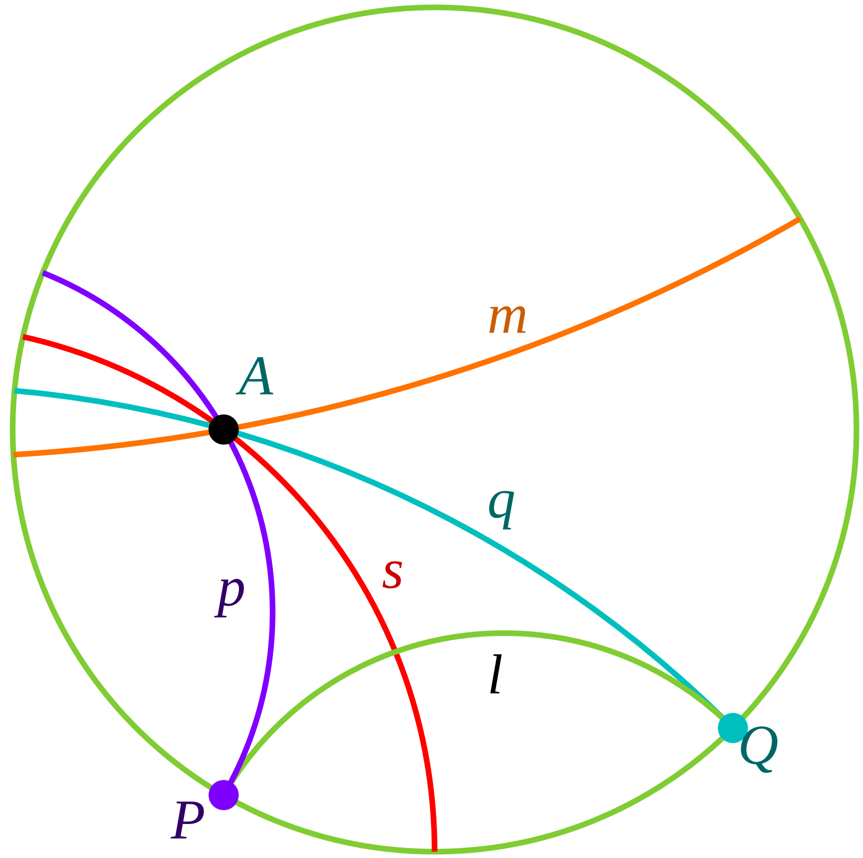
Poincaré's disc model

another
parallel q
to ℓ
through A



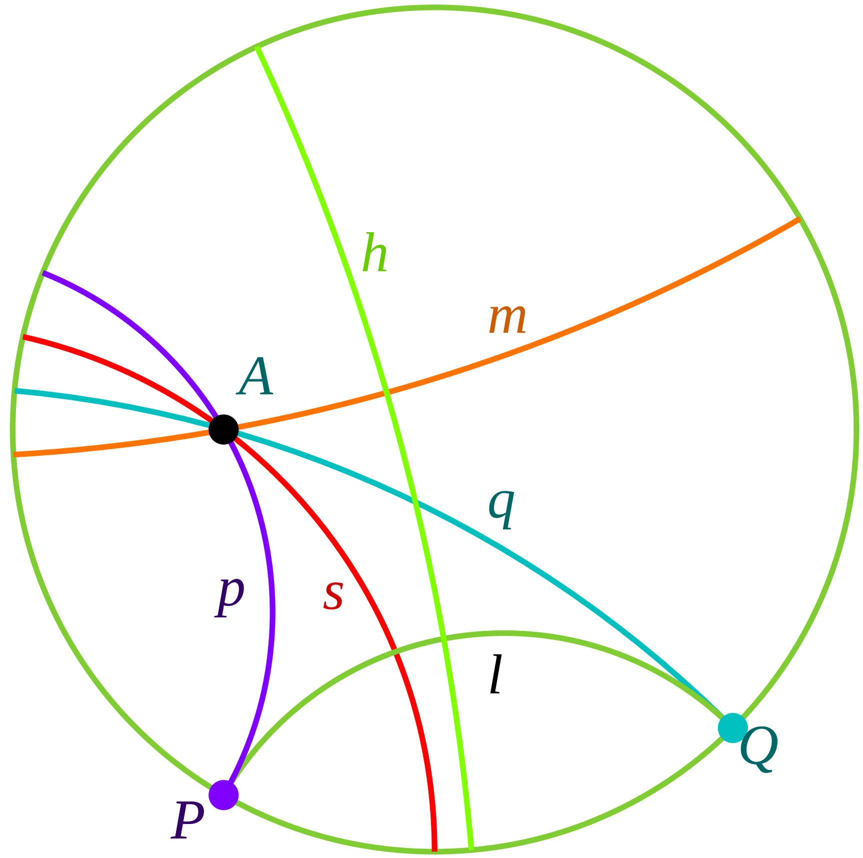
Poincaré's disc model

a **non secant** line m to ℓ through A



Poincaré's disc model

the **common perpendicular** to ℓ and to m



3. pentagrid and dodecagrid

in the **Euclidean spaces**,

the square and the cubic grids

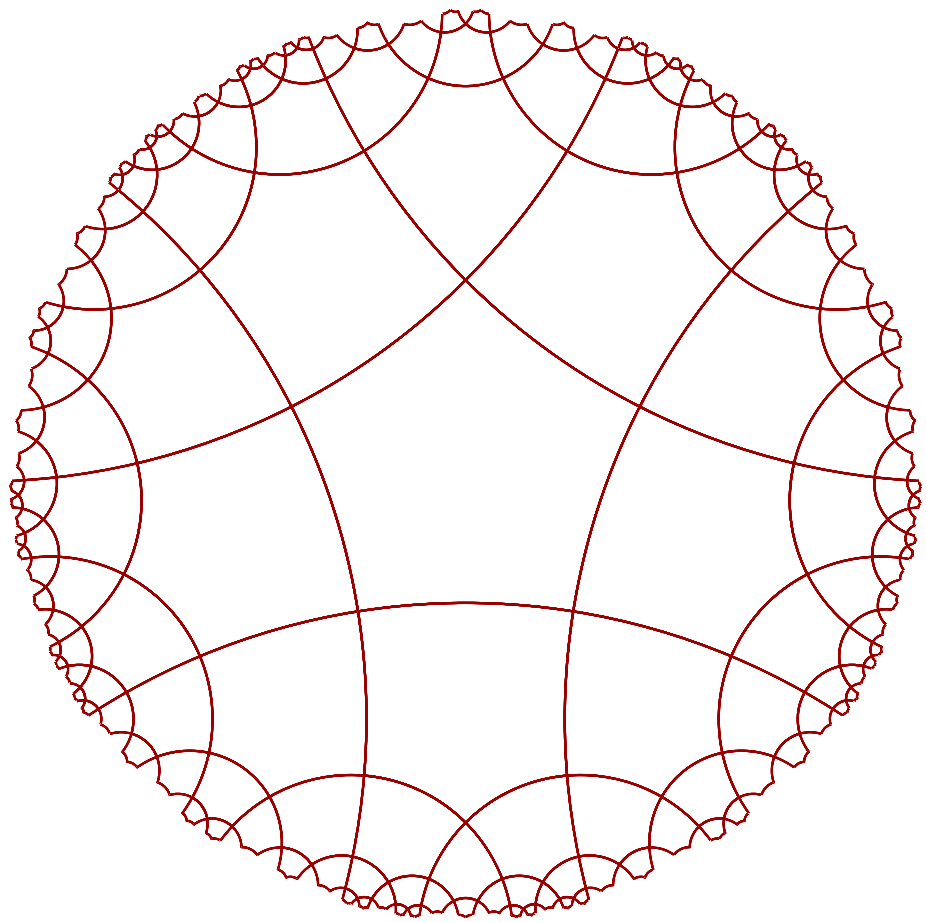
here, this role played by

the **pentagrid**

and the **dodecagrid**

the pentagrid

the simplest **rectangular grid** in the hyperbolic plane



the $3D$ hyperbolic space

extend Poincaré's disc model $\mathcal{P}$
to Poincaré's ball model

points: the unit **open** ball
points at infinity: unit sphere $\mathcal{S}$

planes: trace of
a diametral plane, copy of $\mathcal{D}$,
a sphere orthogonal to $\mathcal{S}$

lines: intersection of planes,
each one in a diametral plane

the $3D$ hyperbolic space

four tessellations
one of them,

based on a **dodecahedron**,

extends the pentagrid to $3D$:

the **dodecagrid**

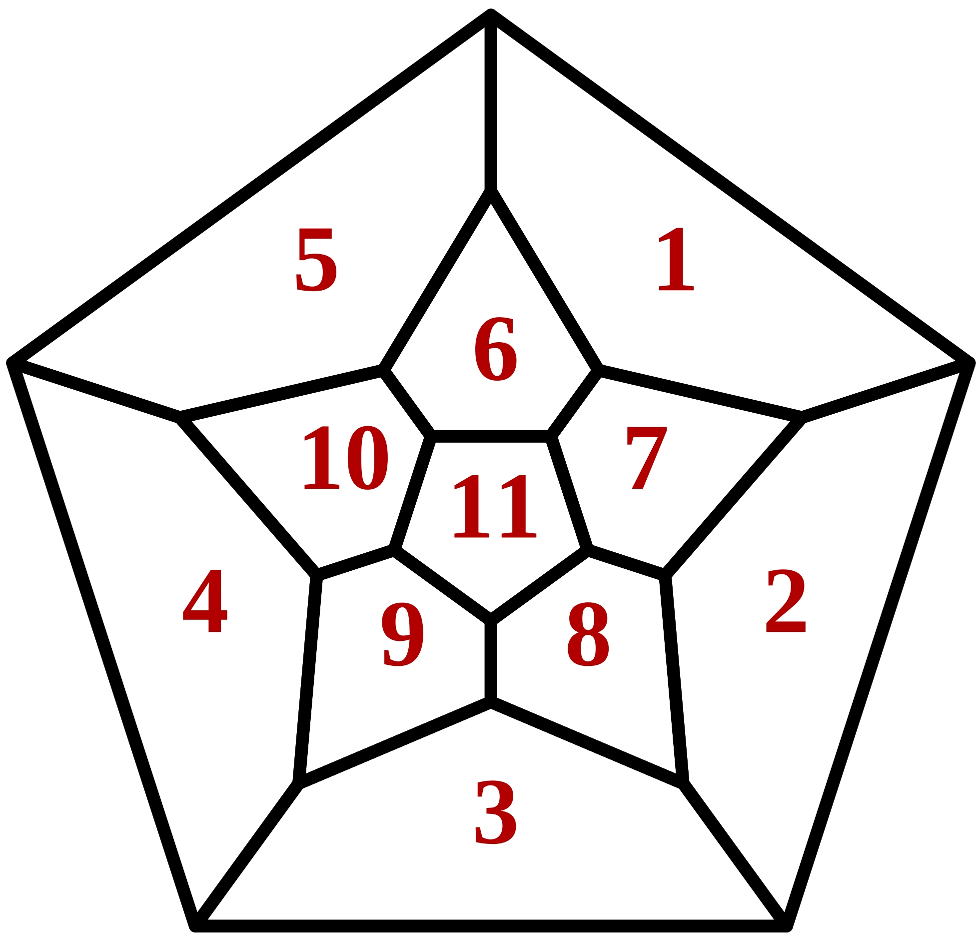
the $3D$ hyperbolic space

important remark:

the trace of the dodecagrid on the plane of one of its faces is the pentagrid

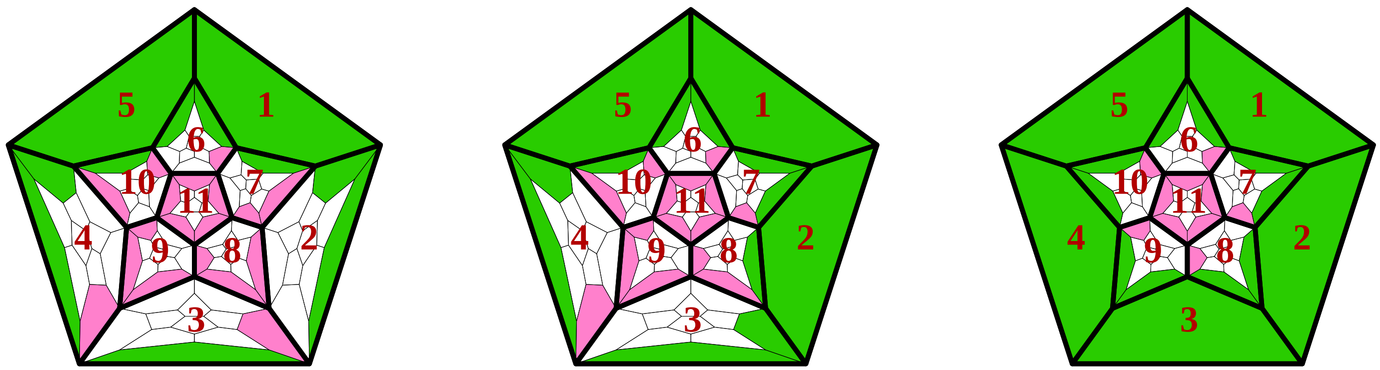
the dodecagrid

projection
onto the plane
of face 0



the dodecagrid

a spanning tree defines
the tiling too,
generationg nodes:



4. CA's in the dodecagrid

CA's in the dodecagrid

neighbours of $\triangle$:

the cells which share a face with $\triangle$

the automaton is deterministic

and rotation invariant:

> if the neighbourhood is changed
> by a motion around $\triangle$
> which keeps orientation,
> the new state is the same

CA's in the dodecagrid

format of a rule

fix a numbering of the faces, then:

$\eta^0\ \eta_0\eta_1\ \eta_2\ \eta_3\ \eta_4\ \eta_5\ \eta_6\ \eta_7\ \eta_8\ \eta_9\ \eta_{10}\ \eta_{11}\ \eta^1$

with:

η^0: current state of the cell

η_i: state of neighbour i,
the other cell sharing face i

η^1: new state of the cell,
after applying the rule

$\eta^0 \ldots \eta_{11}$ is the **context** of the rule

rotation invariance

numbering of the faces fixed,

enumerate all positive motions
leaving Δ globally invariant

to each motion,
associate the word obtained from

the rule once the motion
applied to the cell,
the numbering being still fixed

rotation invariance

the **minimal rotated form**:

the rule whose associated word
is lexicographically the smallest one

hence a test for rotation invariance

lemma

a CA is rotationally invariant
if and only if any pair of its
rules giving rise to the same
minimal rotated context give
the same new state

rotation invariance

enumerating the positive motions:

fix face 0 and face 1, say f_0 and f_1

orientation
$\Rightarrow f_0, f_1$ enough to restore numbering

if $\rho = \rho(\Delta)$, ρ positive motion,
then:
$\rho(f_0)$ is any face: 12 choices
and then $\rho(f_1)$ is any face sharing
an edge with $\rho(f_0)$: 5 choices

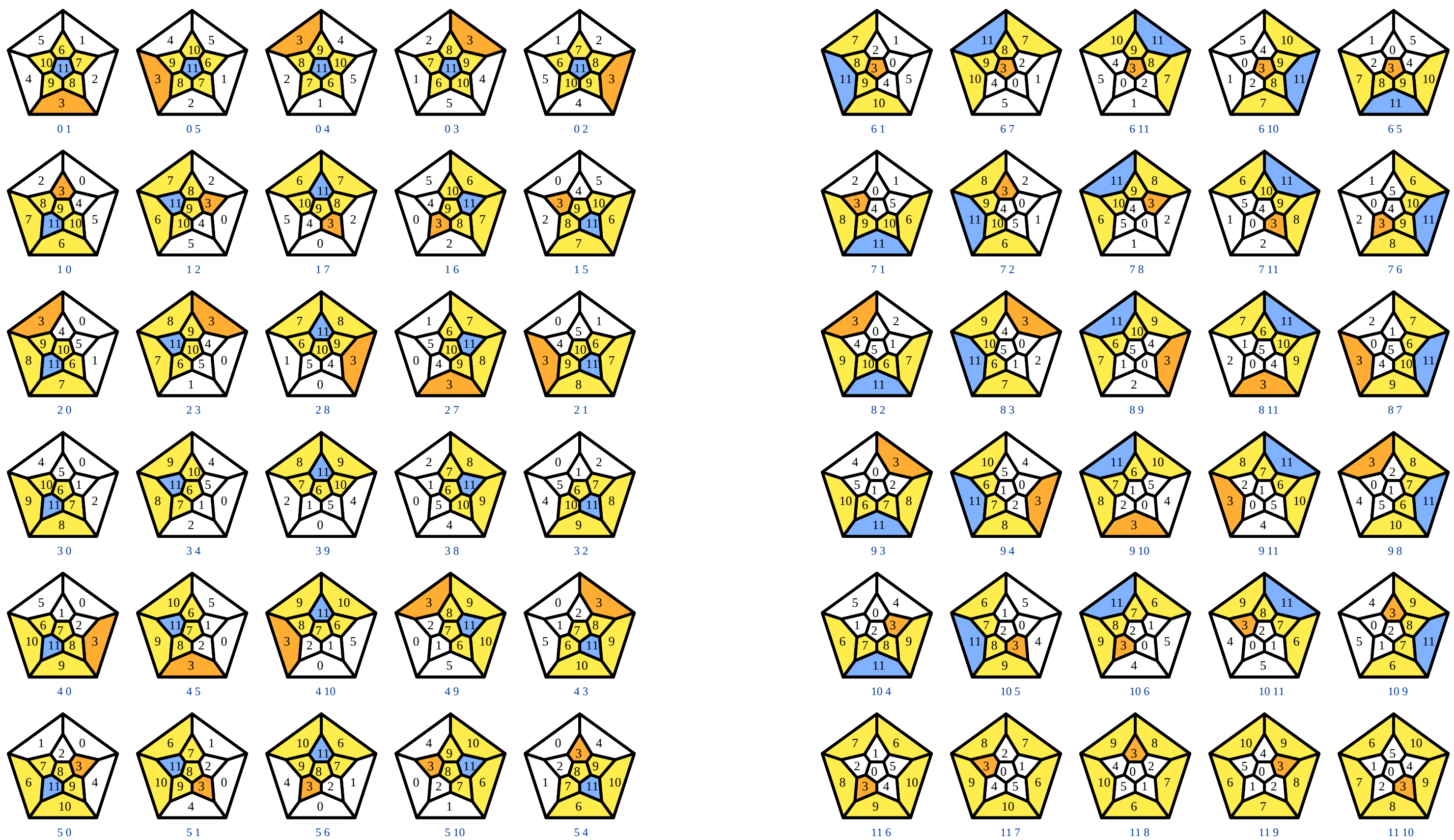
rotation invariance

hence 60 motions leaving
$\triangle$ globally invariant

and we get an easy algorithm to
check whether a CA in the
dodecagrid is rotation invariant

rotation invariance

the enumeration of the positive motions:



rotation invariance

the positive motions leaving Δ globally invariant constitute a group of 60 elements

this group is isomorphic to A_5, the group of permutations on 5 elements with positive signature

A_5 is known to be **simple**, hence:

no nice decomposition,
no easy representation

4. railway simulation

the railway model

circuit in the tiling
which consists of:

tracks
crossings
switches

a unique locomotive runs
over the circuit

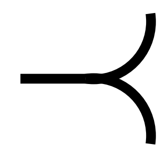
the railway model

the switches

three types of them:

fix

flip-flop



memory

the railway model

working of the switches

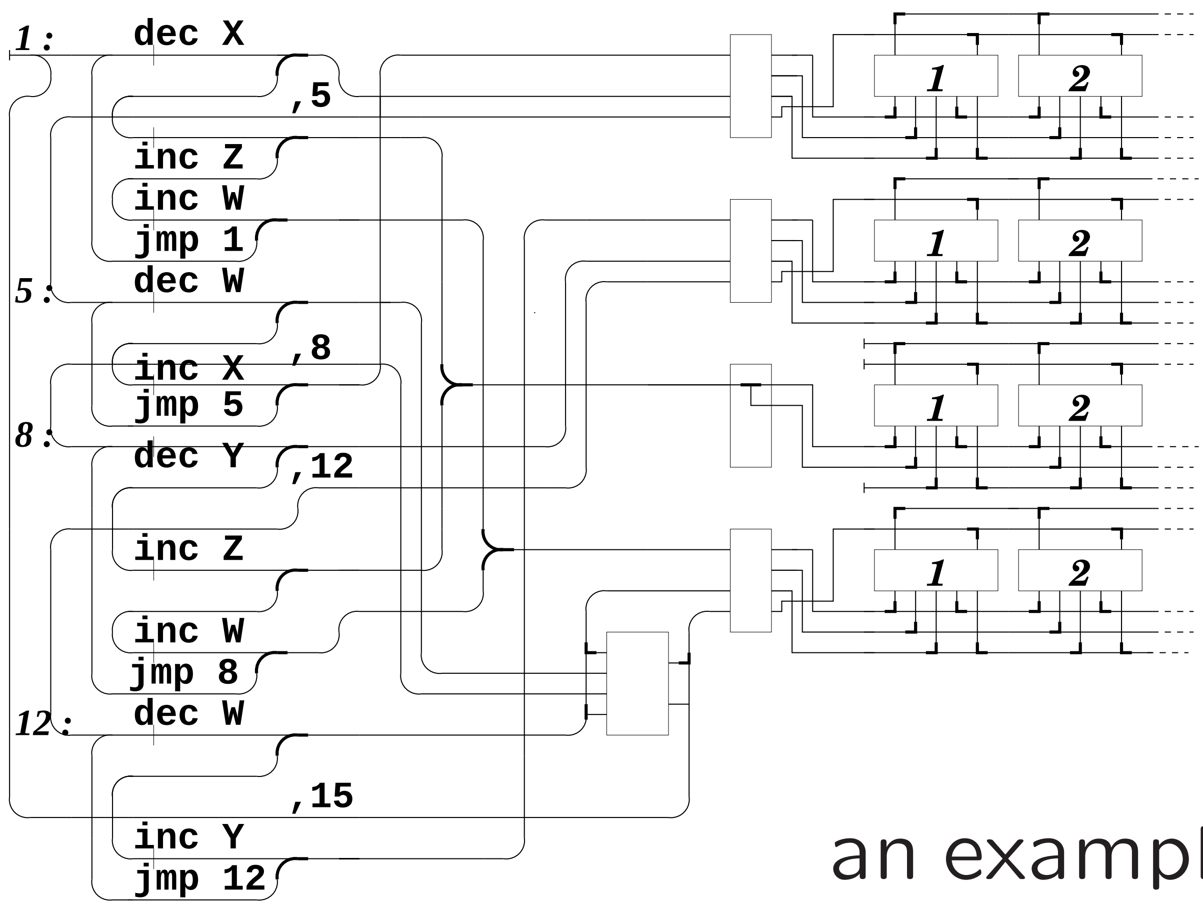
flip-flop:
only active passage,
triggers the change of selection

memory:
selected track = last passive passage

the railway model

it is known that by assembling switches and tracks, a **universal computation** can be simulated by the motion of the locomotive

the railway model



an example

the railway model

implementation of the
example in the pentagrid

5. strong/weak universality in CA's

strong/weak universality:

the issue is the **initial configuration**

strong universality:

the CA has a quiescent state q:
a cell in q with all its neighbours in q remains in q
initial configuration:
all cells are in q except, possibly, finitely many of them

strong/weak universality:

weak universality:

the initial configuration may be **infinite** but, outside a bounded region, it must be regular:

finitely many directions in which the corresponding part of the configuration is invariant under some shift along this direction

6. a universal CA in the dodecagrid with 2 states

our implementations:

previous ones and this one:

mainly in a plane Π_0

use of $3D$:

to replace crossings by bridges
to differentiate the configurations
of the switches

our implementations:

a general remark:

to reduce the number of states
sophisticate the configurations

our implementations:

5 states (MM, 2004):

blank is white
tracks are blue

2-celled locomotive, **green** and **red**
replacing 2 cells of the tracks

neighbouring of the switch centre:
$3D$-decorations

our implementations:

3 states (MM, 2010),

new features:

tracks are white too,
marked with **blue** **milestones**

2-celled locomotive, **blue** and **red**
replacing 2 cells of the tracks

this implementation:

2 states, black and white world

new idea:

one way tracks
again white with black milestones

corollary:
new configuration of the switches

this implementation:

2 states, black and white world

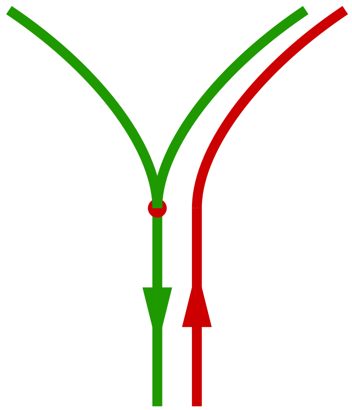
another consequence:

there can no more be a front and a rear for the locomotive

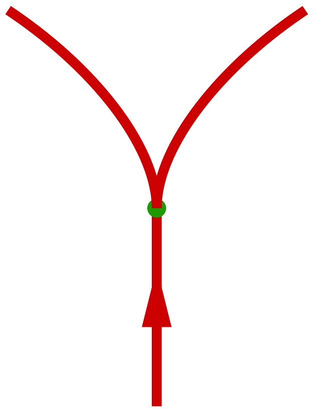
the locomotive is reduced to a unique black cell: the **particle**

our implementation

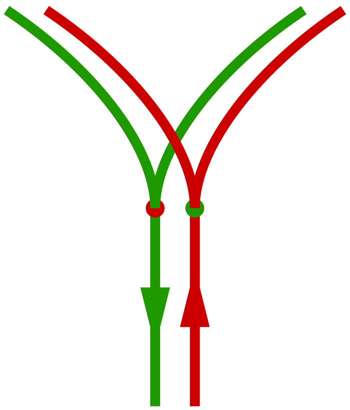
the new configuration of the switches:



fixed switch



flip-flop switch



memory switch

our implementation

a remark:

introducing one way traks
introduces a new distinction:

active and **passive** switches

fixed switch: **passive**

flip-flop: **active**

memory switch: combination of
both types

our implementation:

representation of parts of the circuit

tracks

bridges

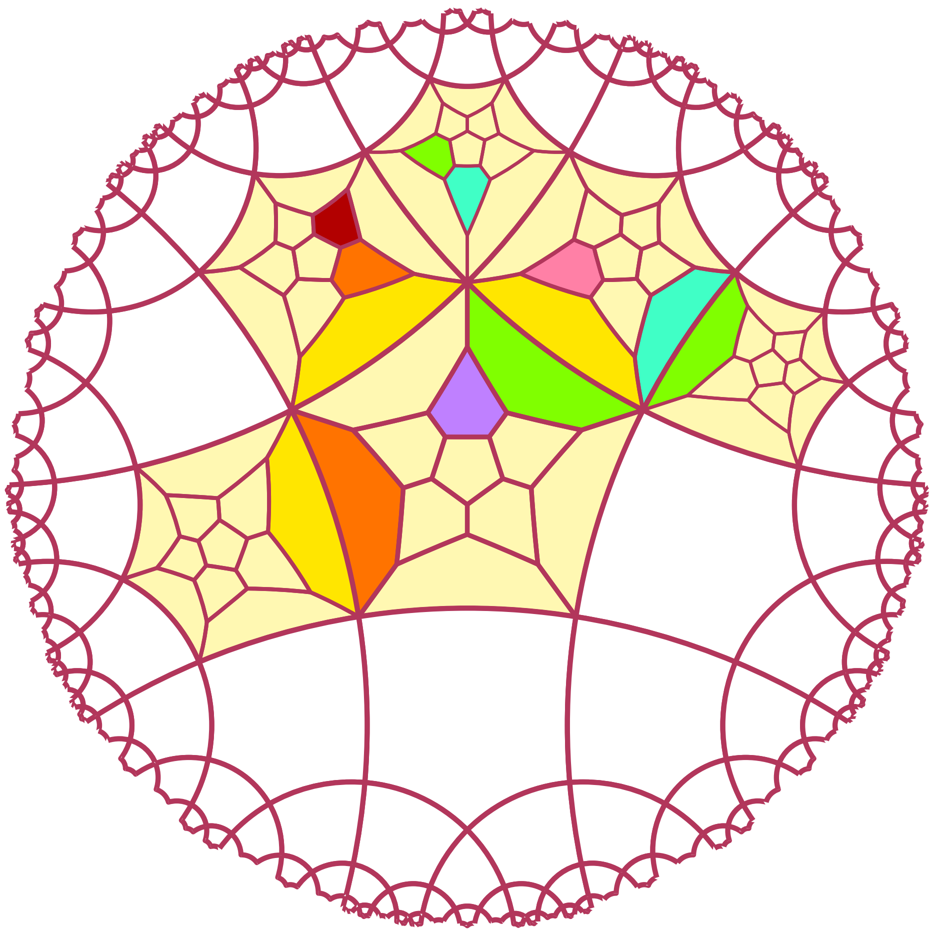
fixed switches

flip-flop switches

memory switches

our implementation:

representations by a **special projection**



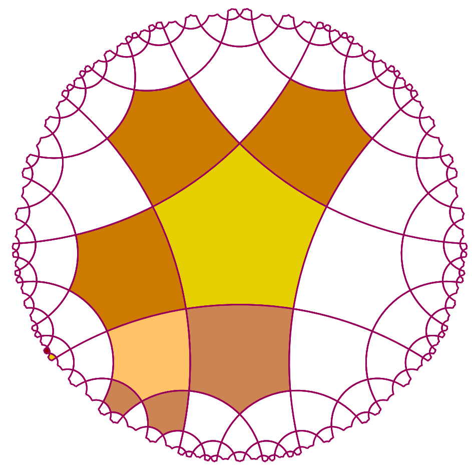
our implementation:

tracks

the idea:

milestones look
like catenaries

same basis for
the return track
which is on the
other half-space



our implementation:

tracks

two kinds of tracks
to implement the circuit:

vertical and *horizontal* tracks

verticals:

they follow branches of a tree

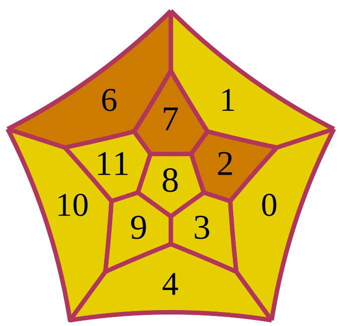
horizontals:

they follow levels of a tree

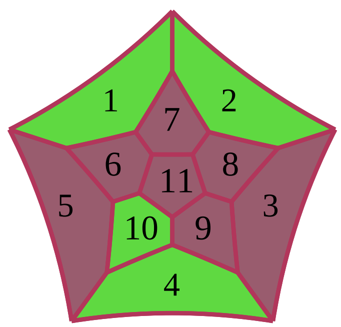
our implementation:

tracks

this induces two kinds of
elements for the tracks:

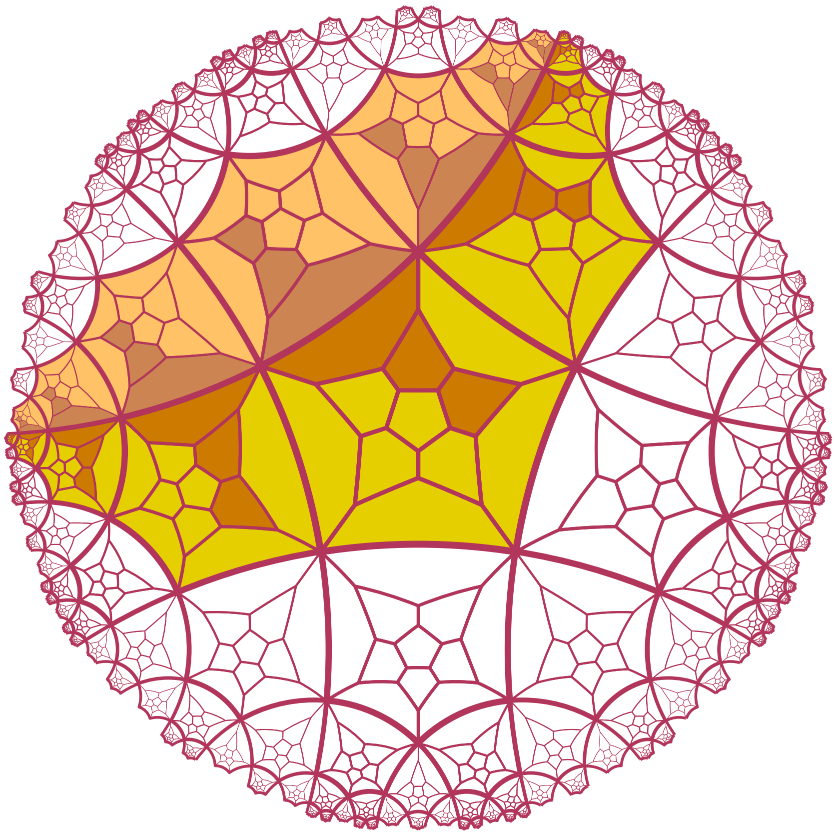


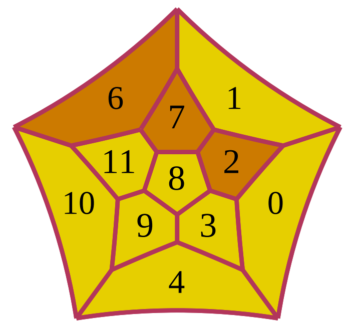
straight element

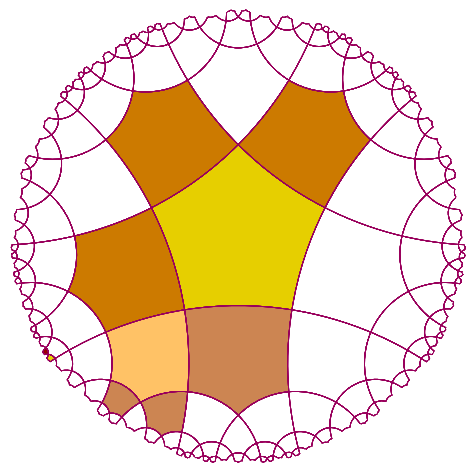


corner

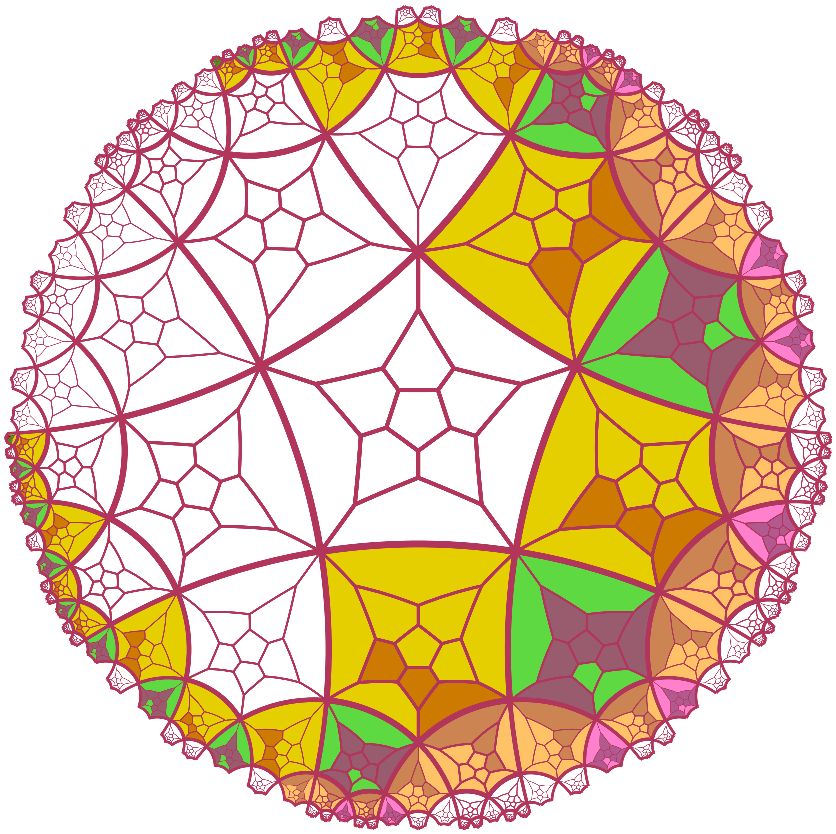
vertical tracks

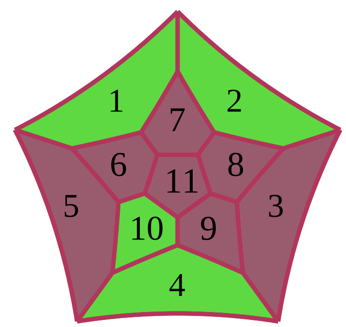






horizontal tracks





our implementation:

bridges

they replace crossings

we may assume the meeting of two vertical segments

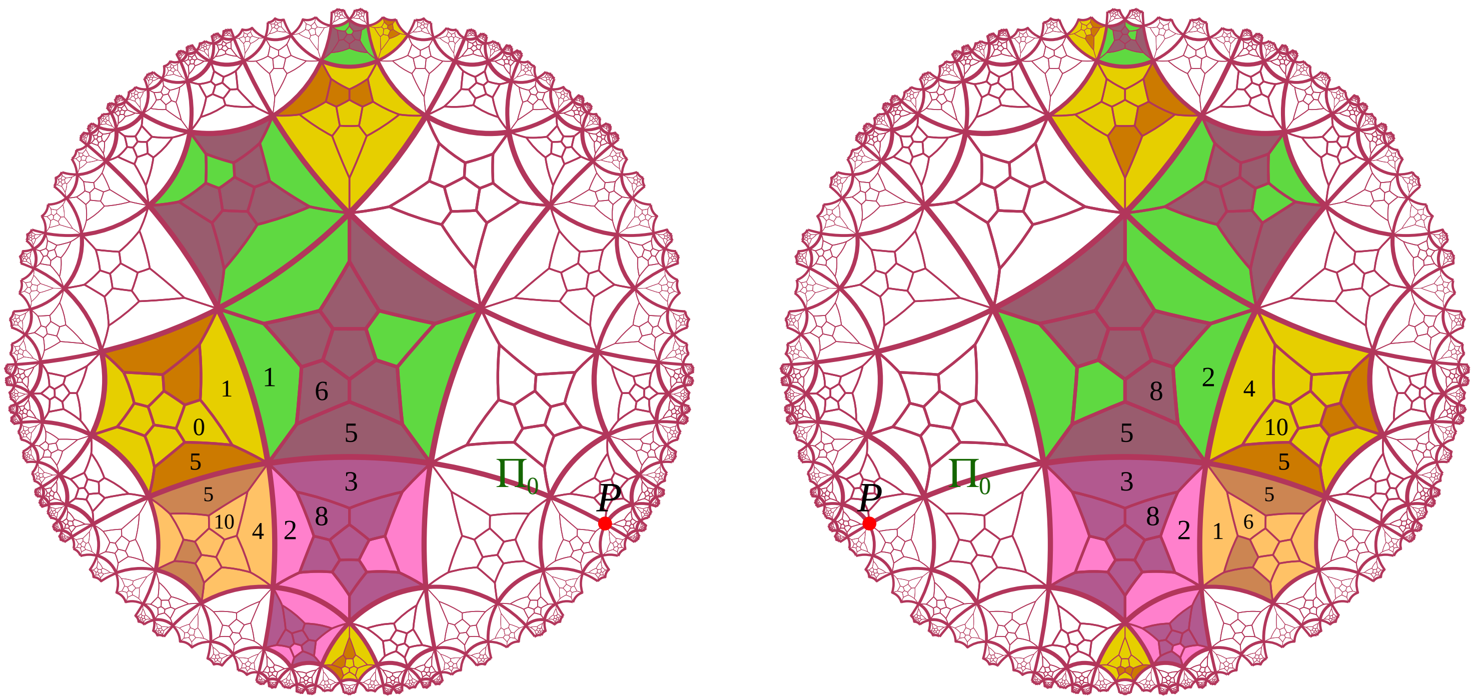
one segment is unchanged

the other makes use of two bridges:

- a bridge for one way
- the symmetric one for the return track

bridges

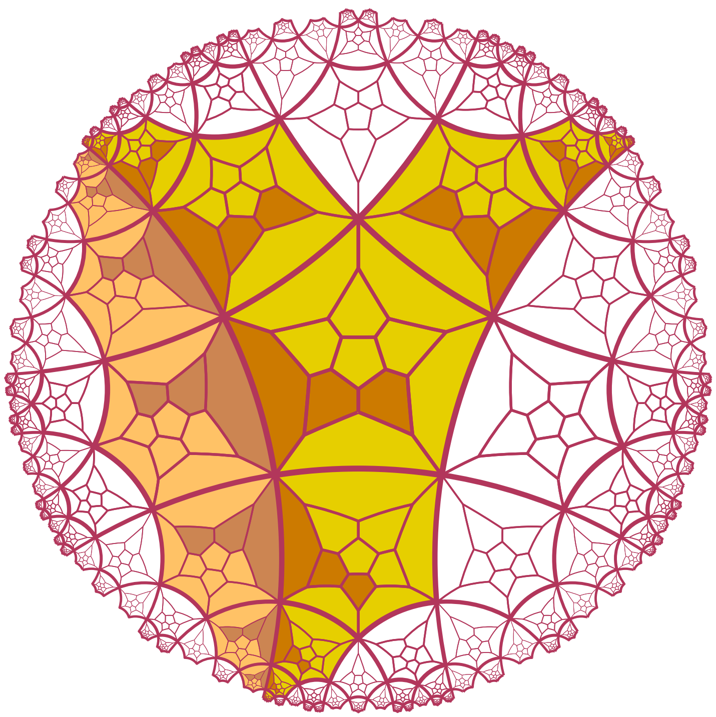
illustration of their ends

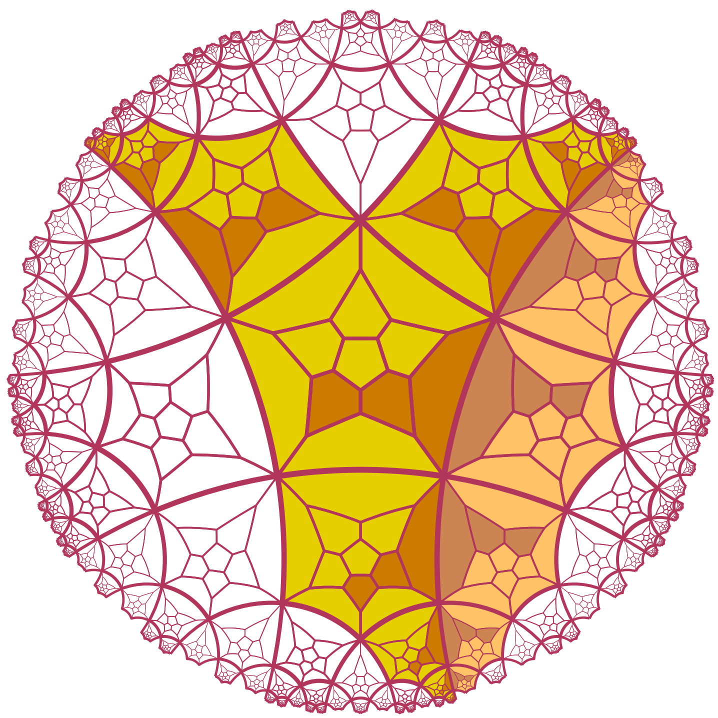


our implementation:

fixed switches

implemented with straight
elements only

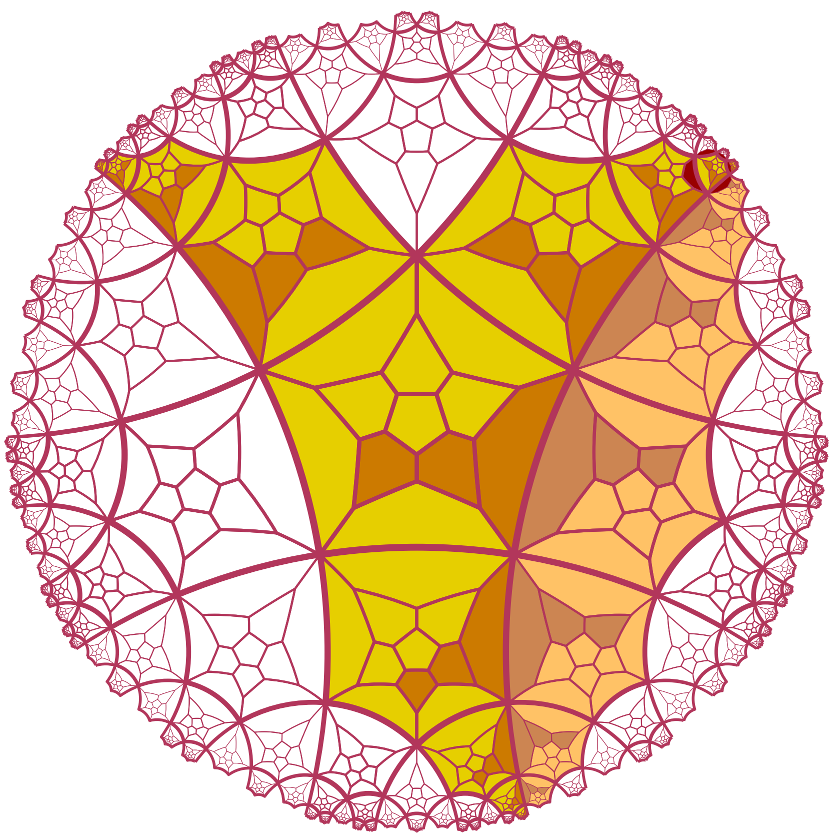




our implementation:

fixed switches

motion of the particle, selected track:

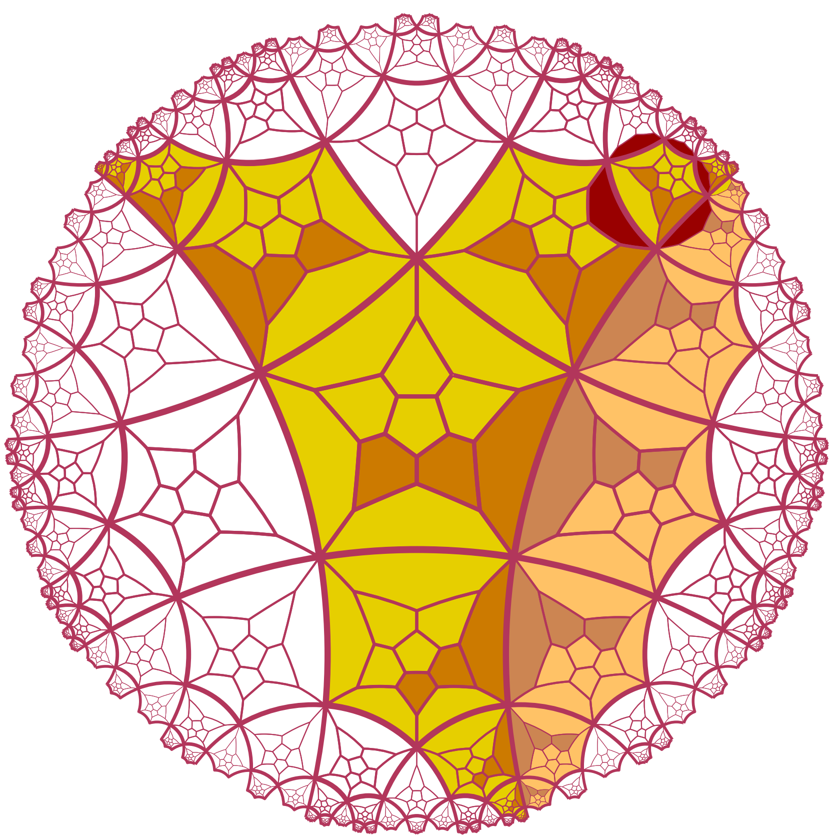


time 1

our implementation:

fixed switches

motion of the particle:

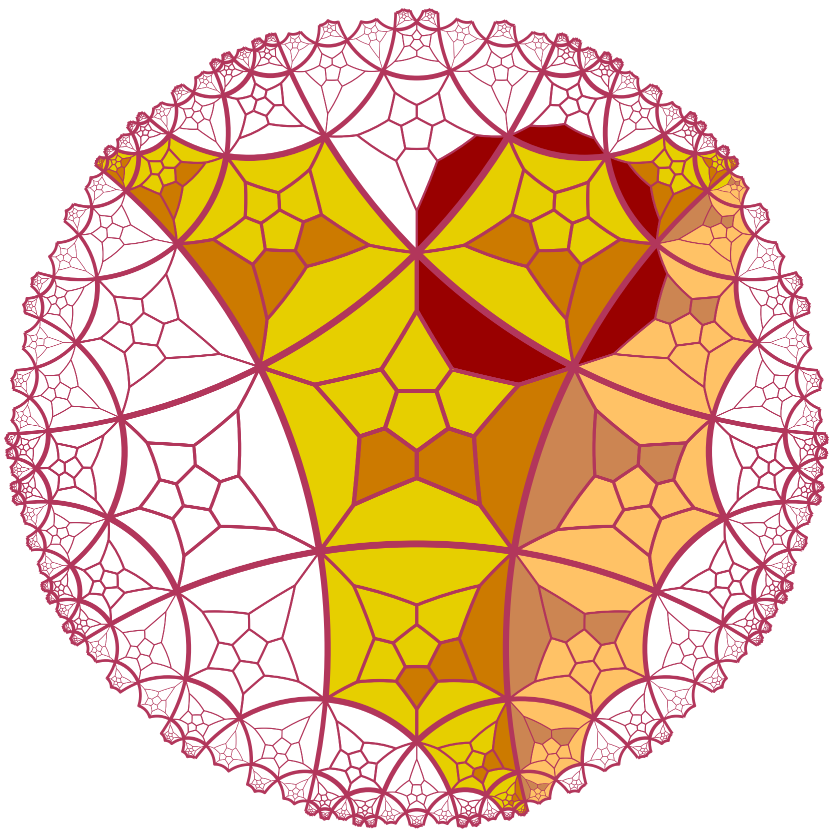


time 2

our implementation:

fixed switches

motion of the particle:

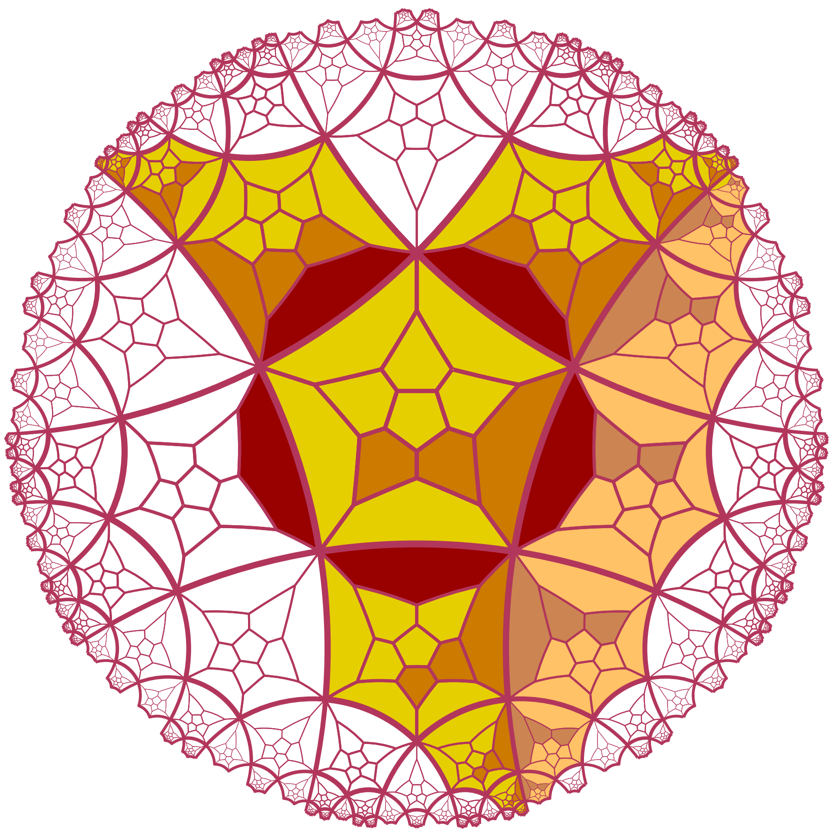


time 3

our implementation:

fixed switches

motion of the particle:

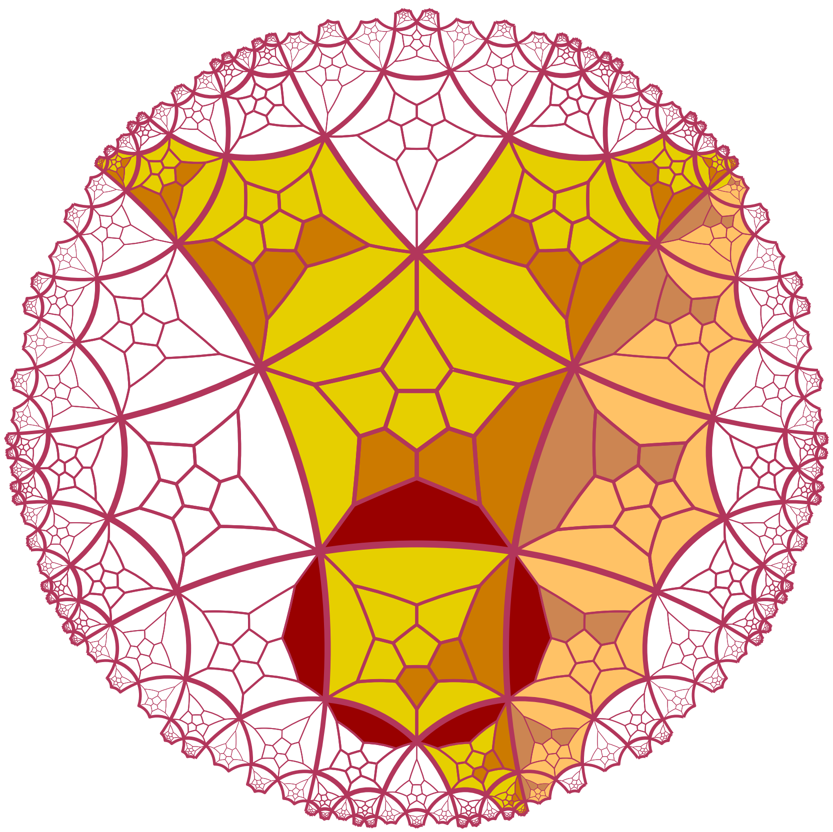


time 4

our implementation:

fixed switches

motion of the particle:

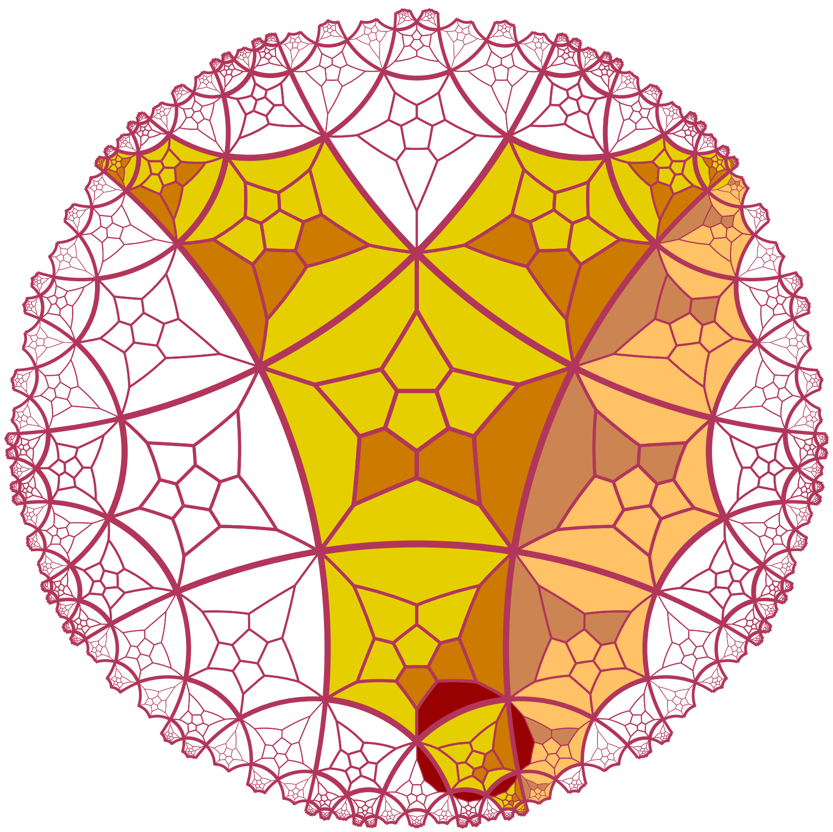


time 5

our implementation:

fixed switches

motion of the particle:

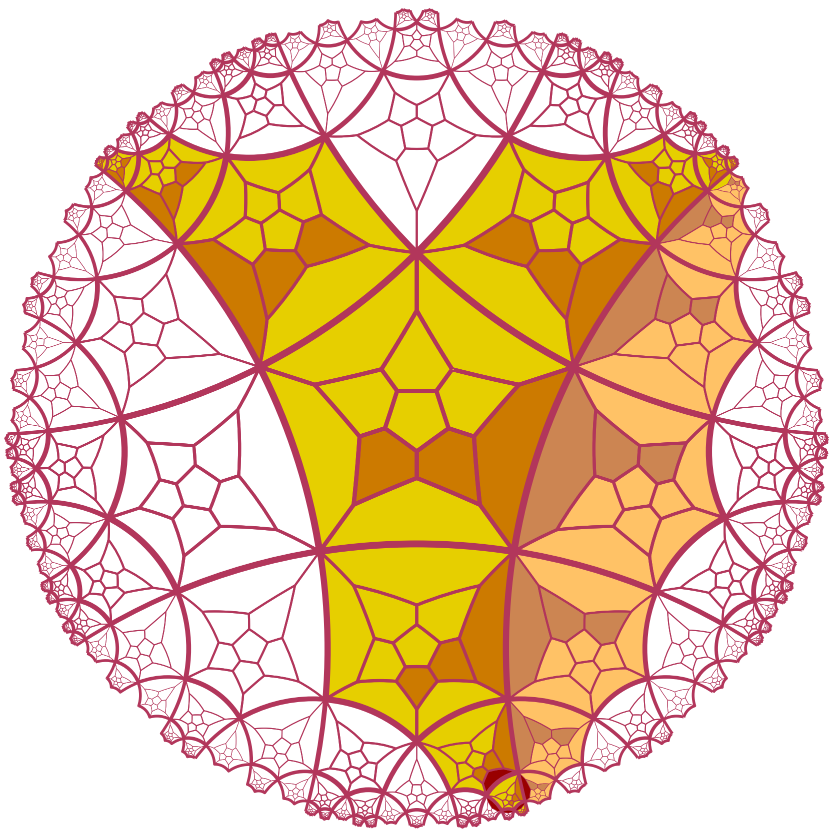


time 6

our implementation:

fixed switches

motion of the particle:

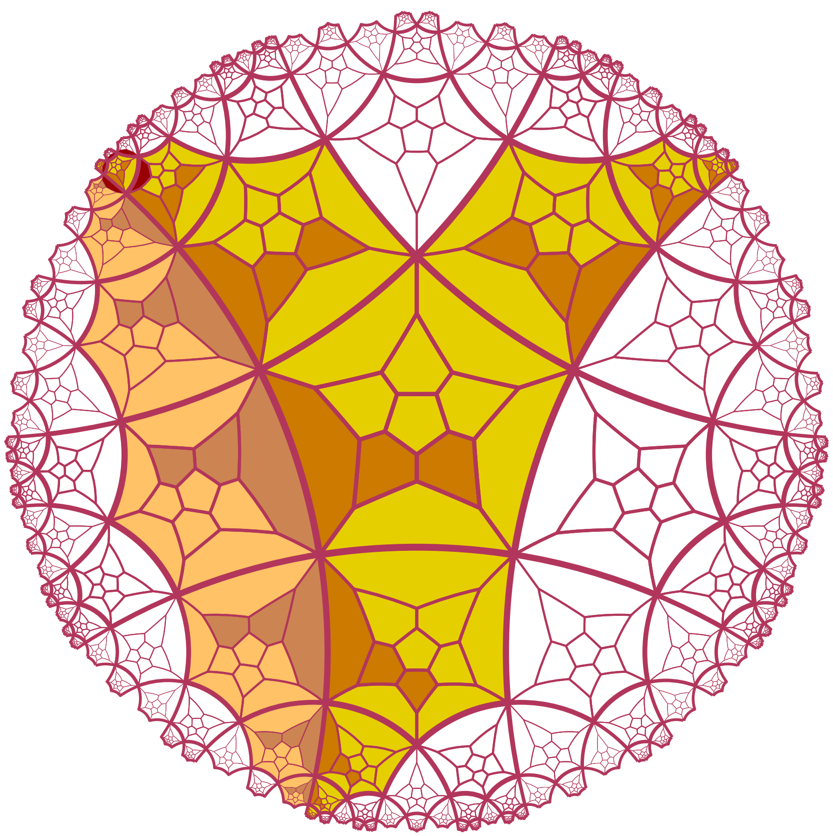


time 7

our implementation:

fixed switches

motion from the non selected track:

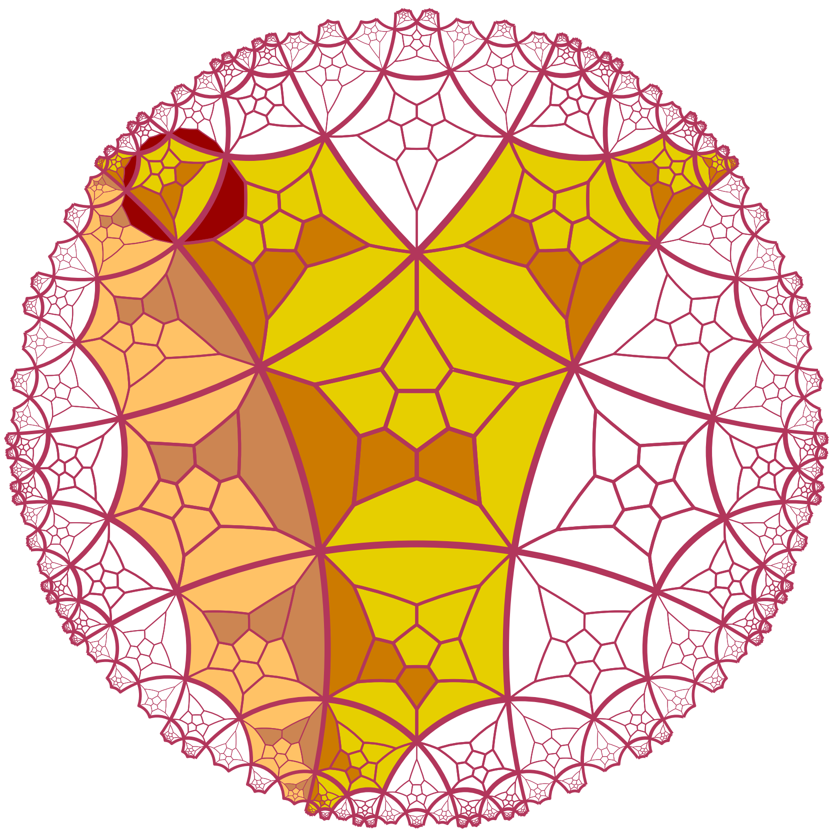


time 1

our implementation:

fixed switches

motion of the particle:

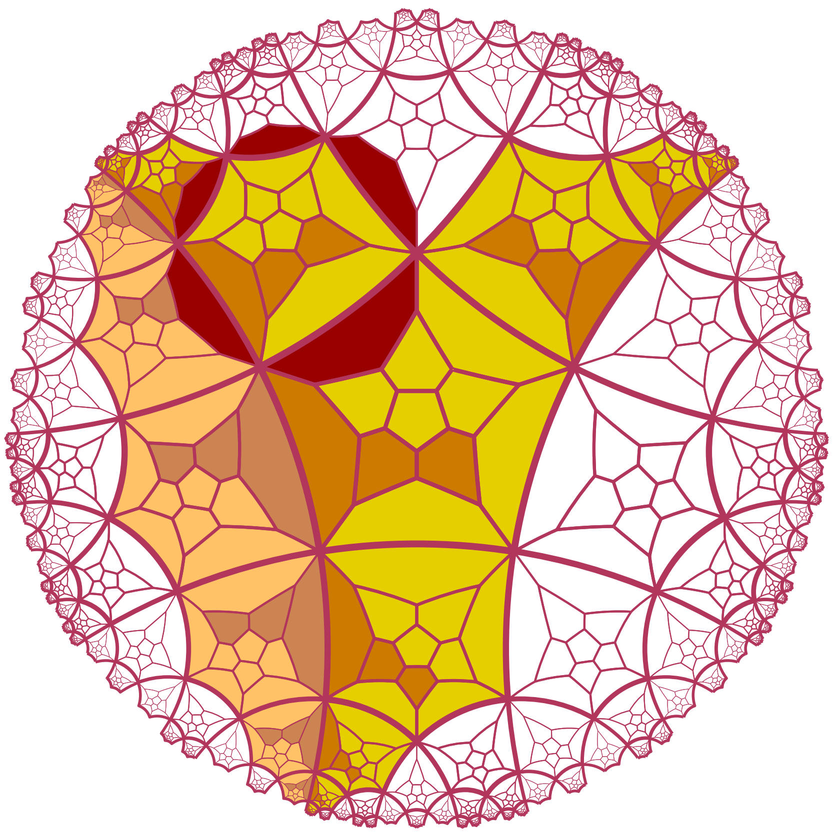


time 2

our implementation:

fixed switches

motion of the particle:

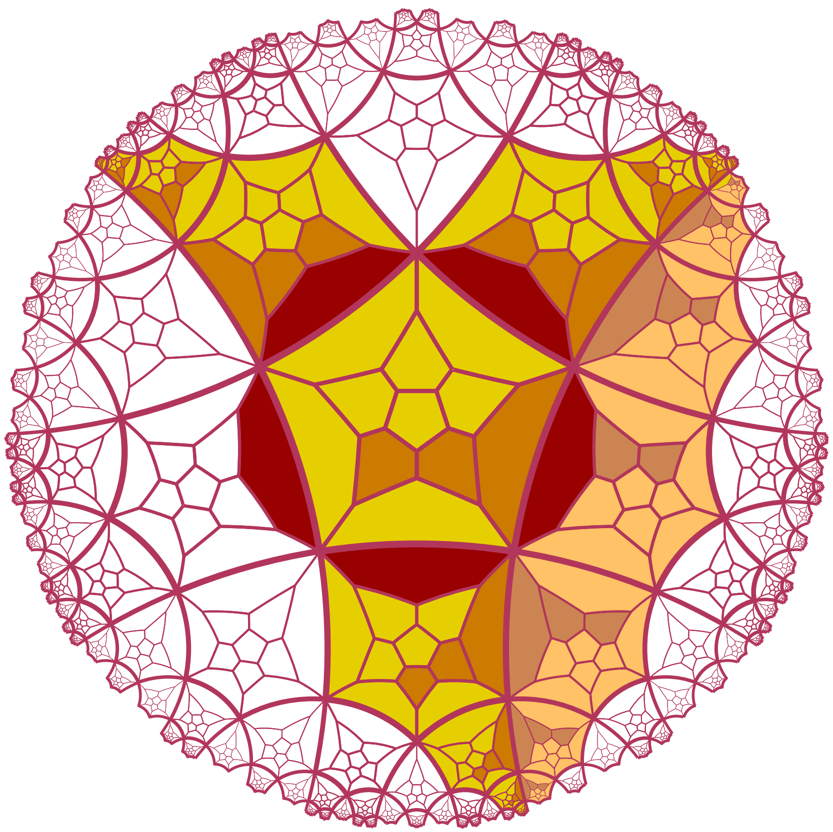


time 3

our implementation:

fixed switches

motion of the particle:

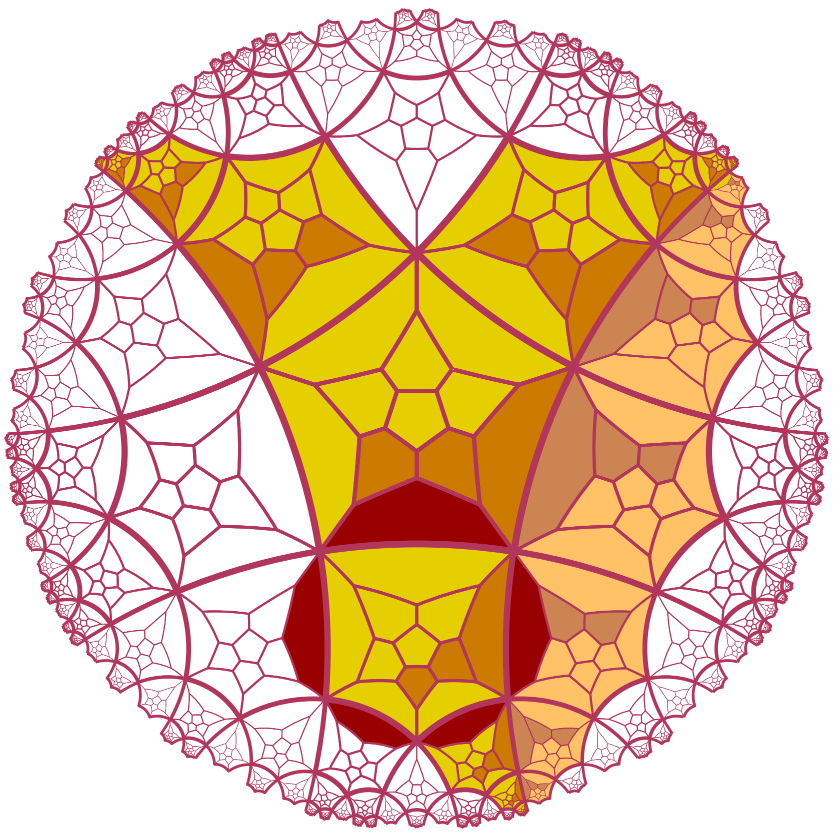


time 4

our implementation:

fixed switches

motion of the particle:

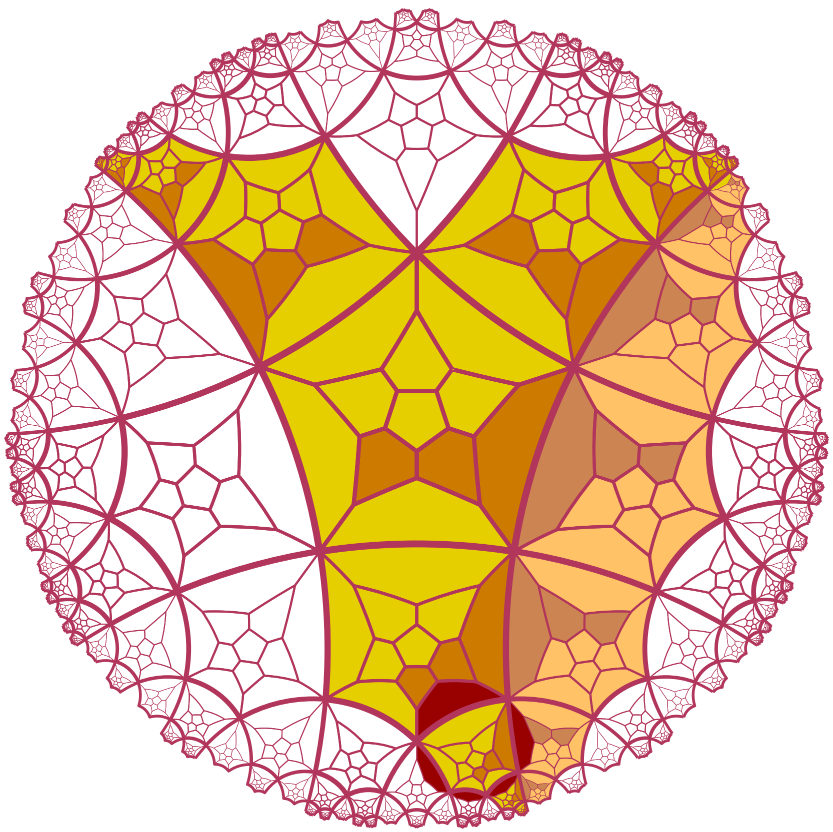


time 5

our implementation:

fixed switches

motion of the particle:

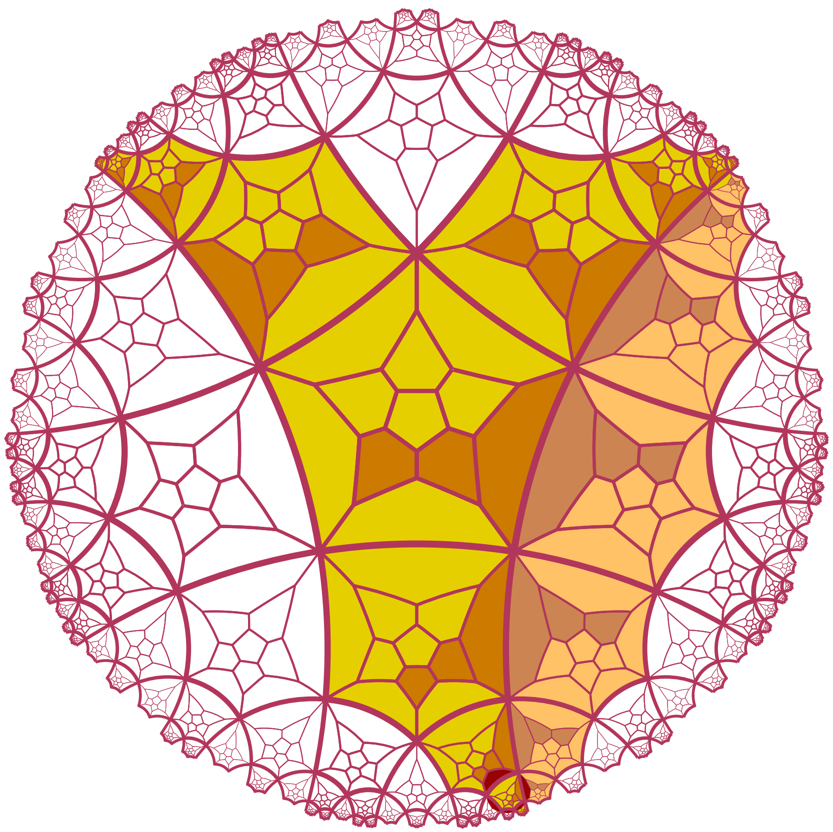


time 6

our implementation:

fixed switches

motion of the particle:

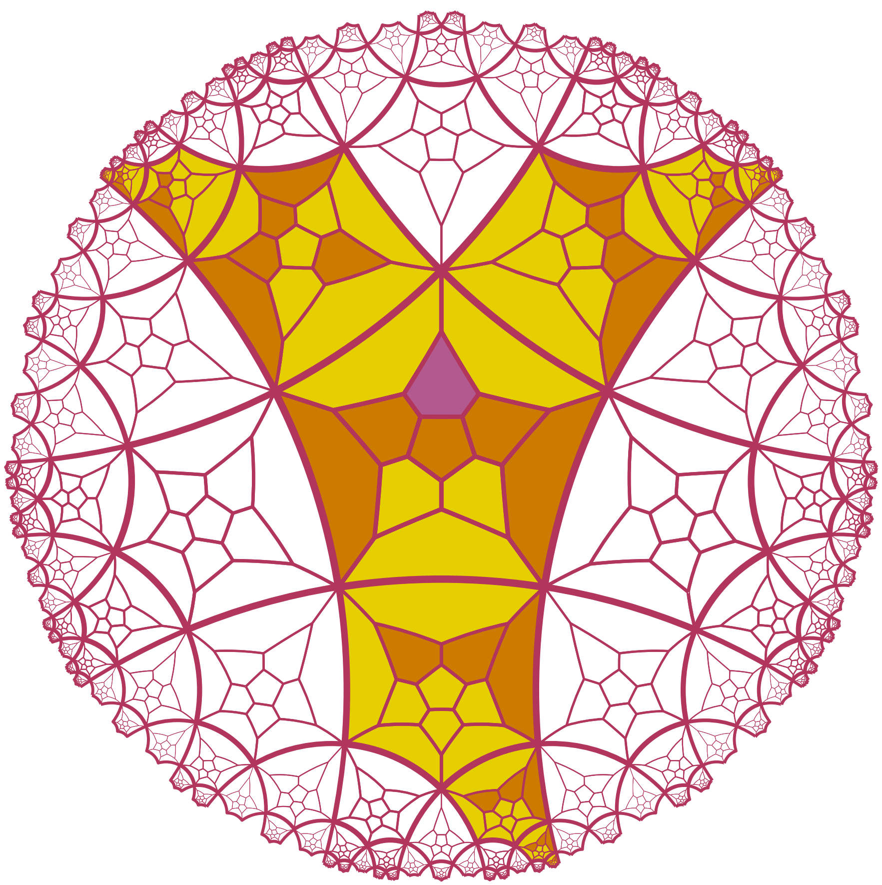


time 7

our implementation:

flip-flop switches

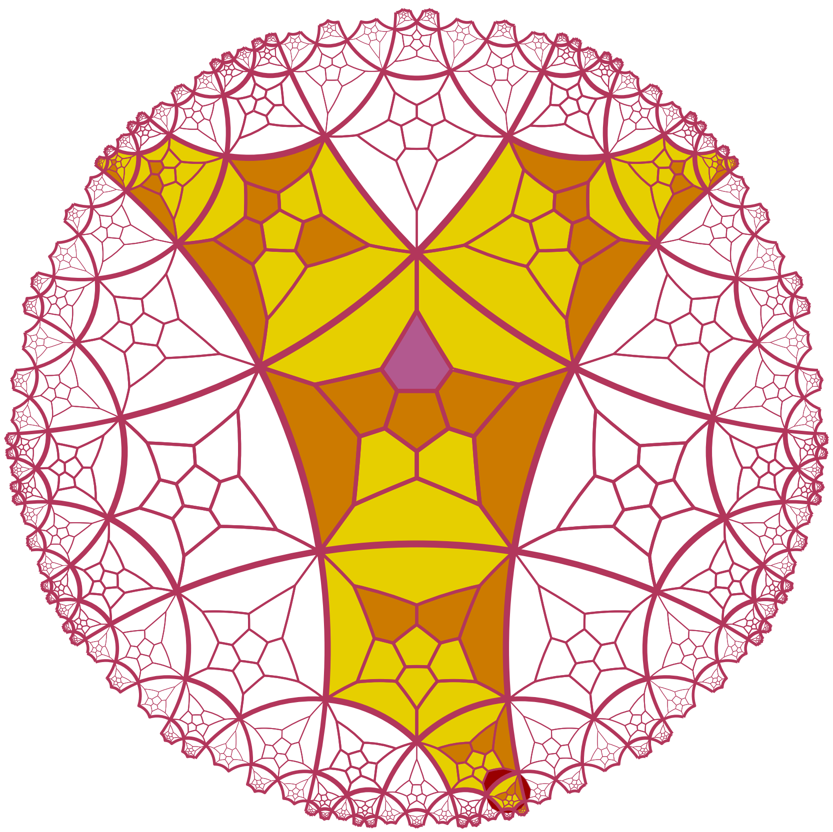
again, implemented with straight elements only



our implementation:

flip-flop switches

motion of the particle:

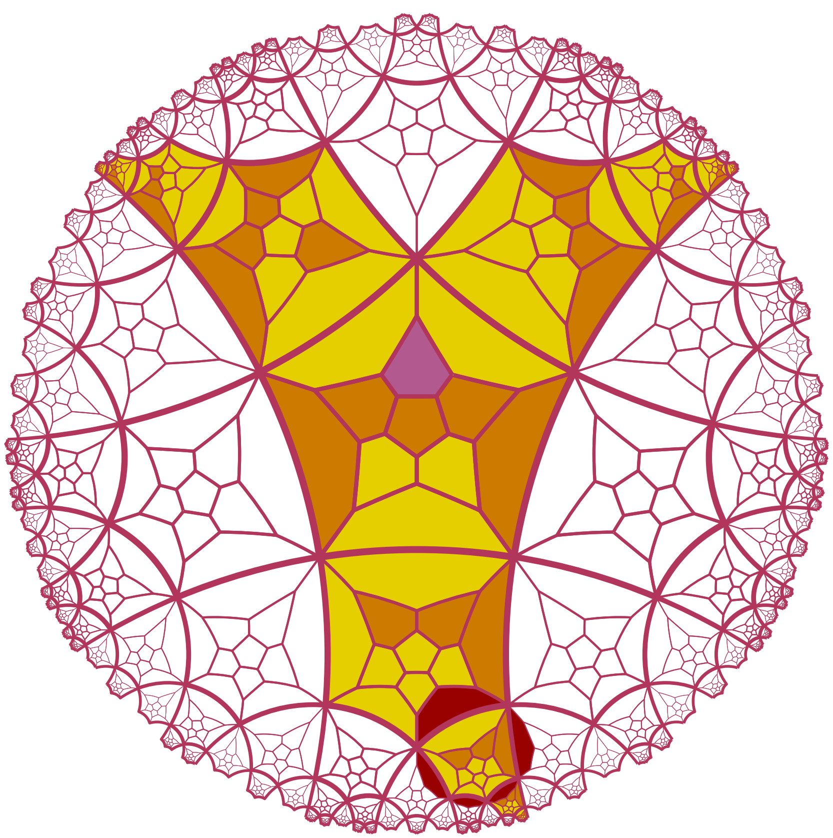


time 1

our implementation:

flip-flop switches

motion of the particle:

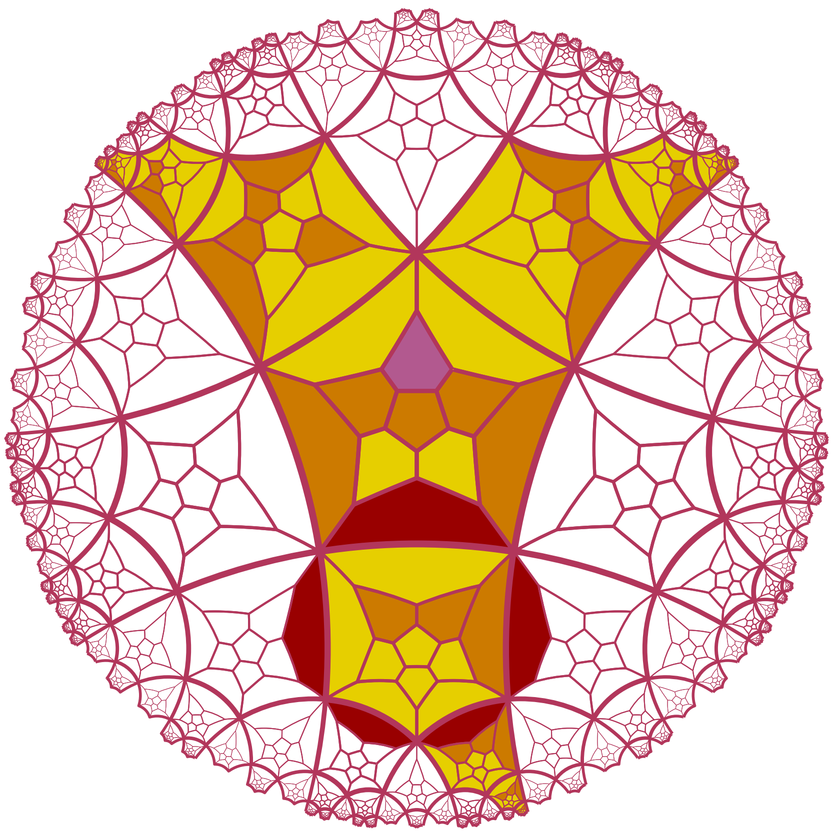


time 2

our implementation:

flip-flop switches

motion of the particle:

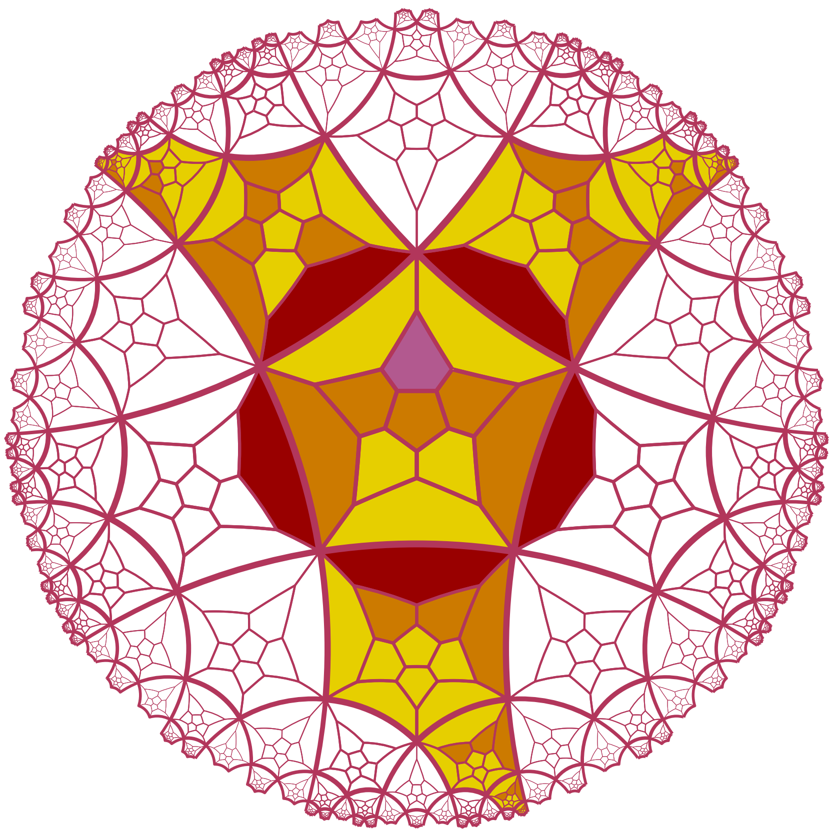


time 3

our implementation:

flip-flop switches

motion of the particle:

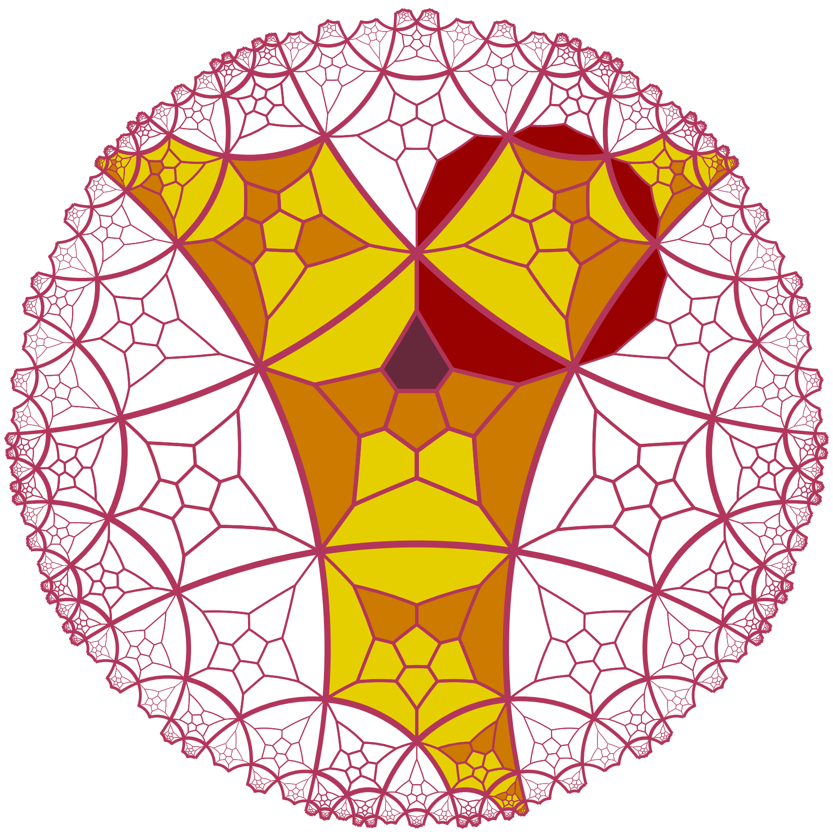


time 4

our implementation:

flip-flop switches

motion of the particle:

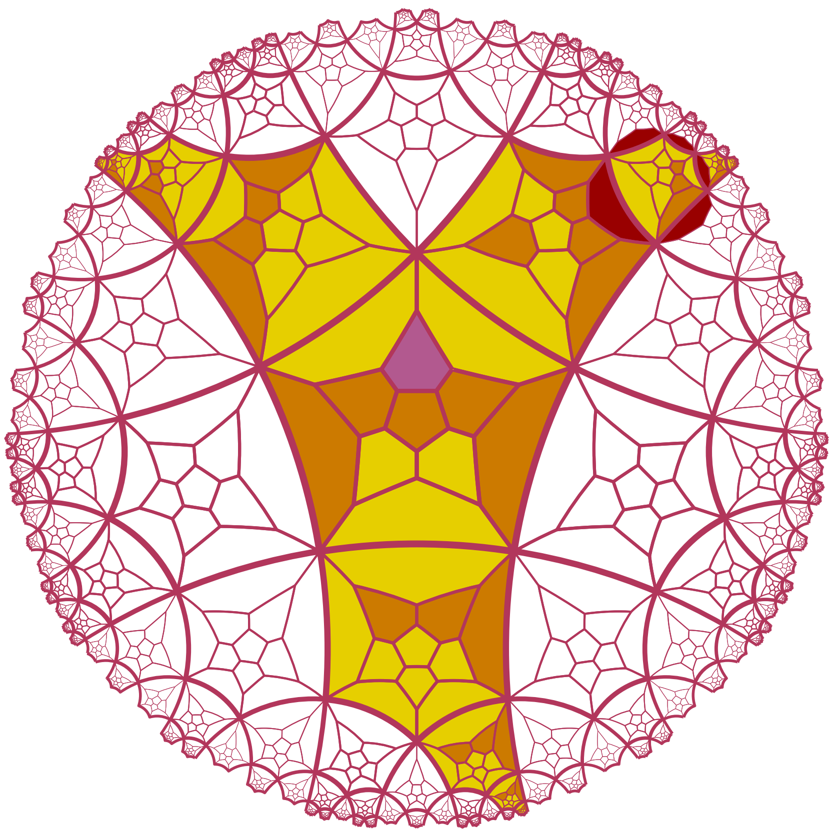


time 5

our implementation:

flip-flop switches

motion of the particle:

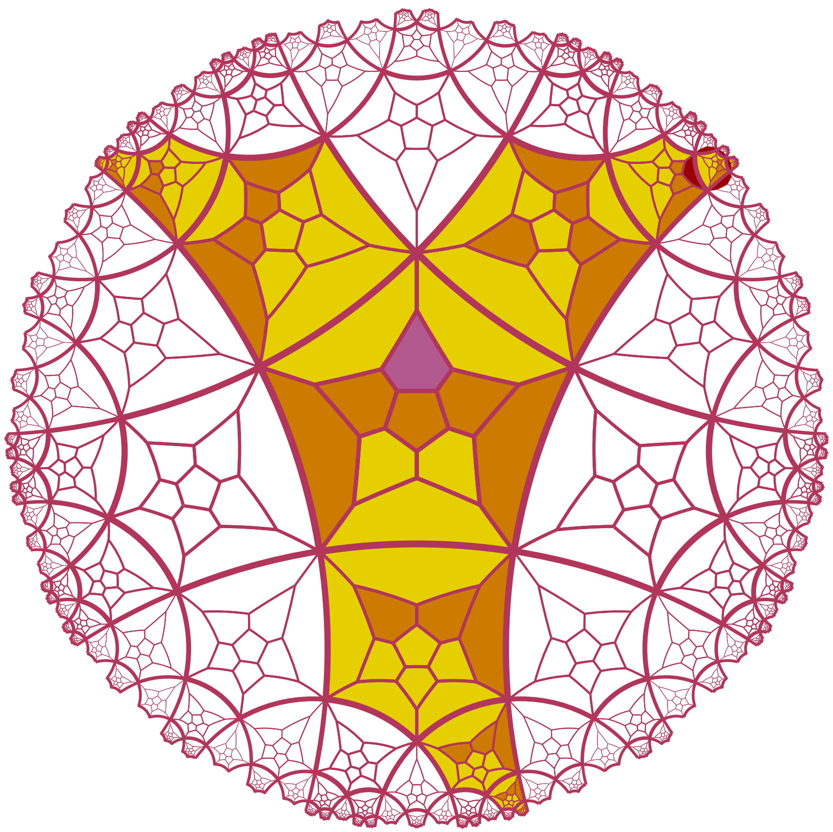


time 6

our implementation:

flip-flop switches

motion of the particle:



time 7

our implementation:

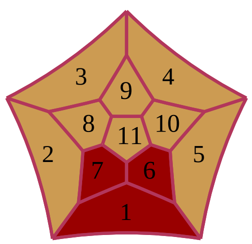
flip-flop switches

of course, if the particle comes again, the flip-flop will switch to the other position

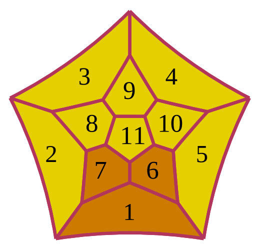
our implementation:

flip-flop switches

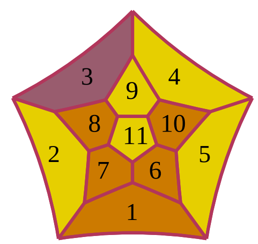
the controller and the sensors:



cell 11



cell 12



cell 13

sensor left

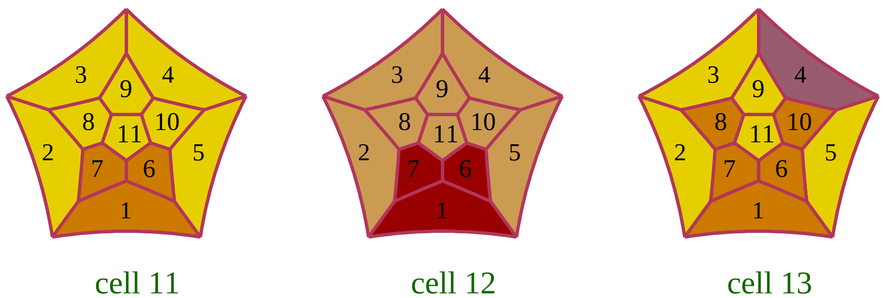
sensor right

controller

our implementation:

flip-flop switches

the controller and the sensors:



sensor left

sensor right

controller

our implementation:

memory switches

this time, both types of switches:

a passive one and an active one

but both are connected

the ‘passive’ switch is somehow
active

the ‘active’ switch is somehow
passive

our implementation:

memory switches

indeed:

for the **passive memory switch**:

when the particle comes from
the **non-selected** track,

this triggers a change in the
active memory switch

our implementation:

memory switches

for the **active memory switch**:

when the particle crosses it,

there is **no change** in the switch,

a change occurs only when an
appropriate signal is sent by
the **passive** memory switch

below we detail:

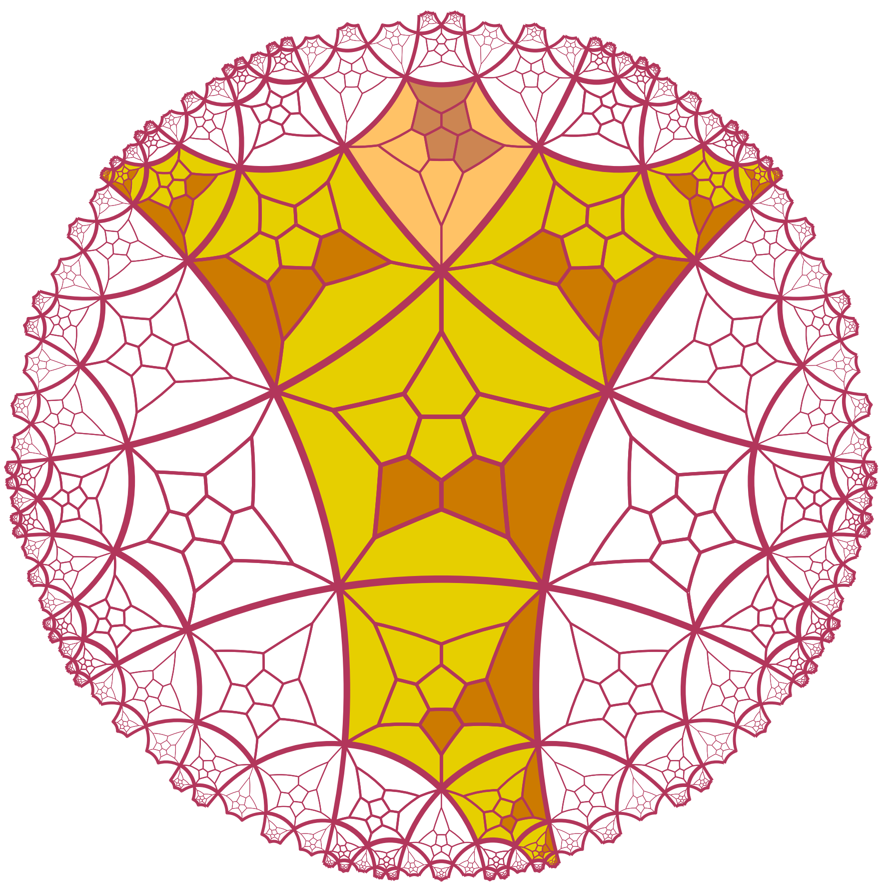
crossing a passive memory switch

the signal to the active switch

our implementation:

passive memory switches

again, implemented with straight elements only



our implementation:

passive memory switches

here, note a new element,

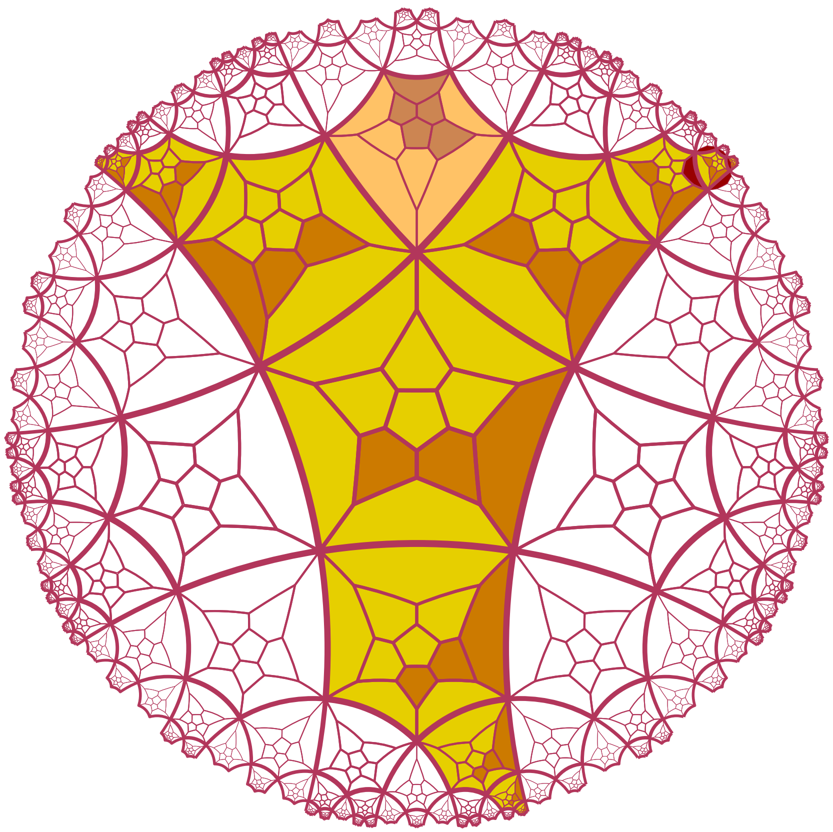
the **passive controller**

it detects whether the particle
arrives through the non-selected track

our implementation:

passive memory switches

motion of the particle:

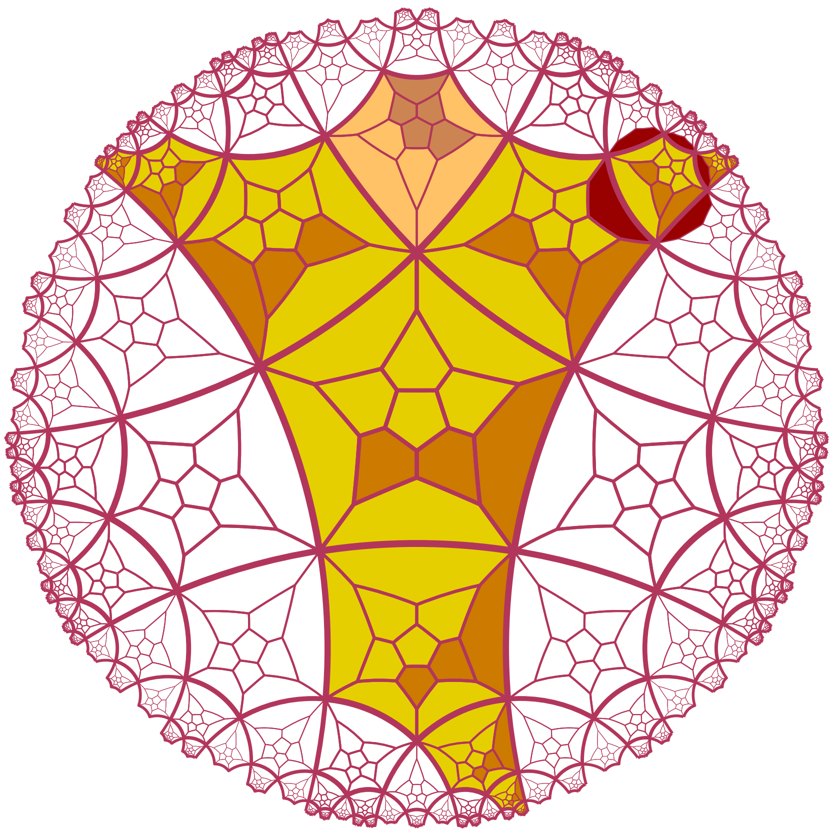


time 1

our implementation:

passive memory switches

motion of the particle:

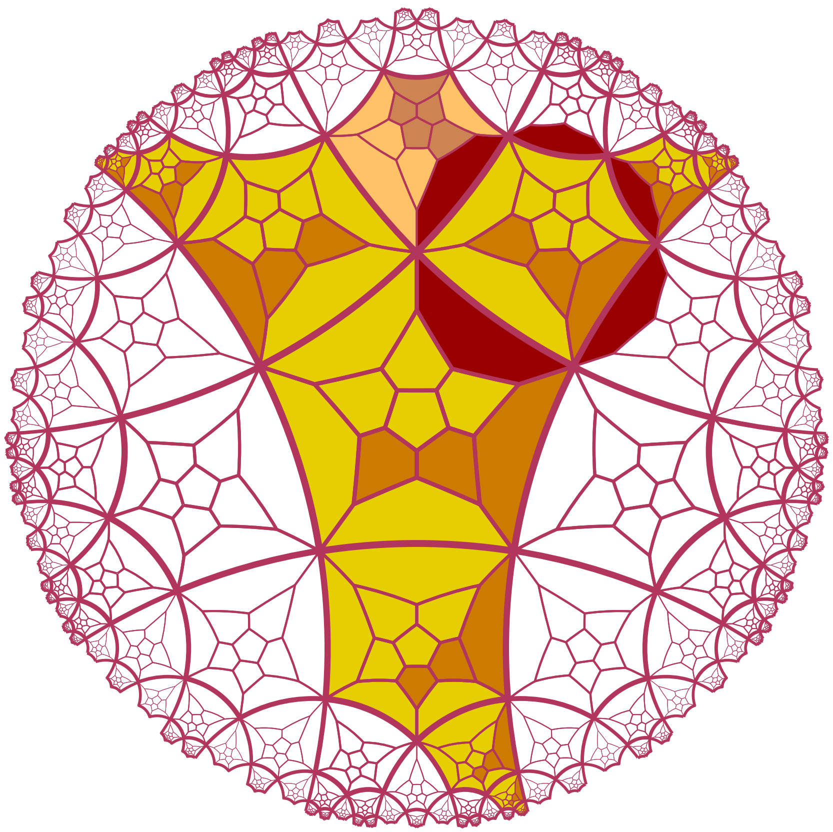


time 2

our implementation:

passive memory switches

motion of the particle:

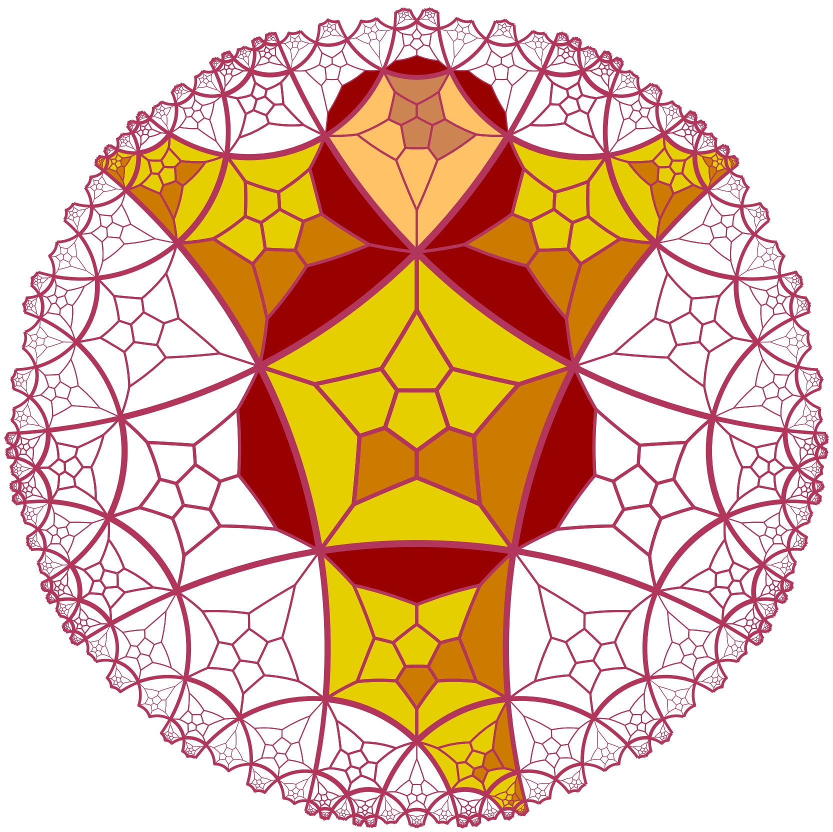


time 3

our implementation:

passive memory switches

motion of the particle:

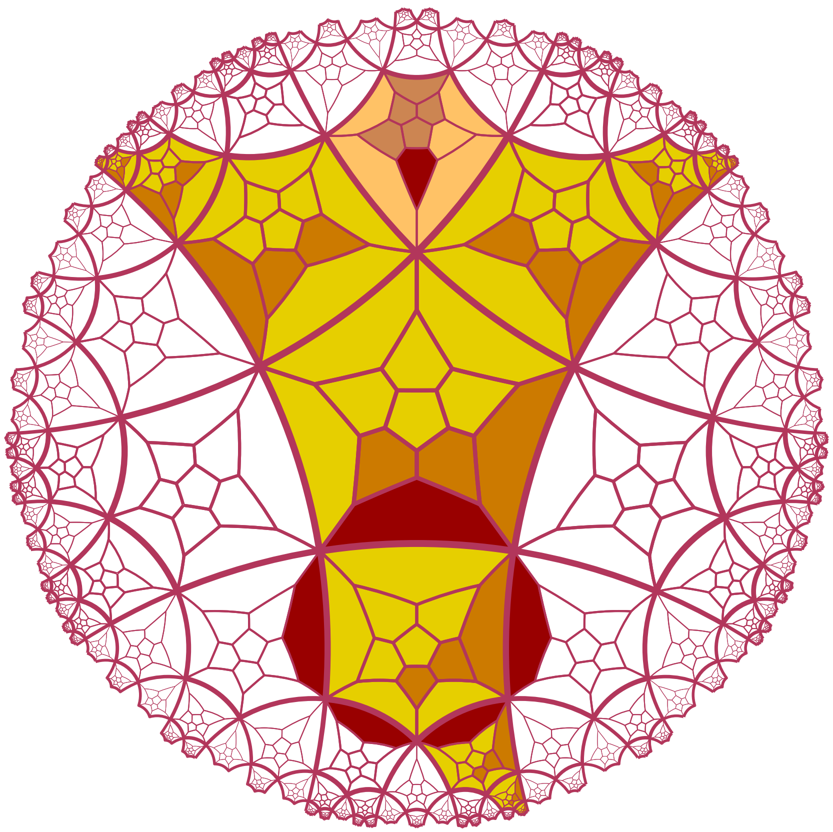


time 4

our implementation:

passive memory switches

motion of the particle:

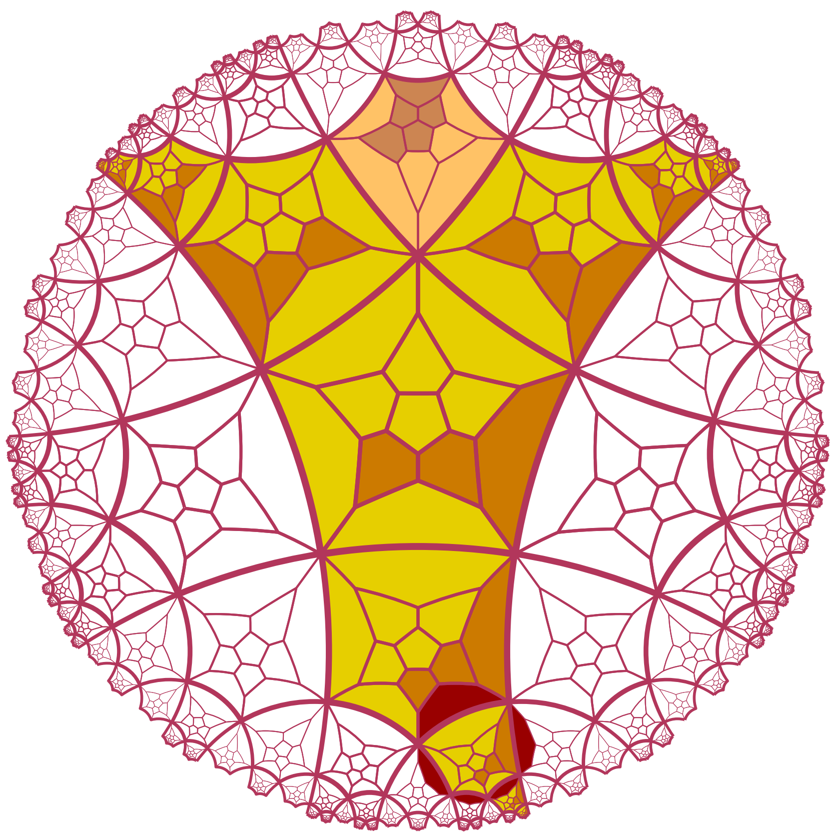


time 5

our implementation:

passive memory switches

motion of the particle:

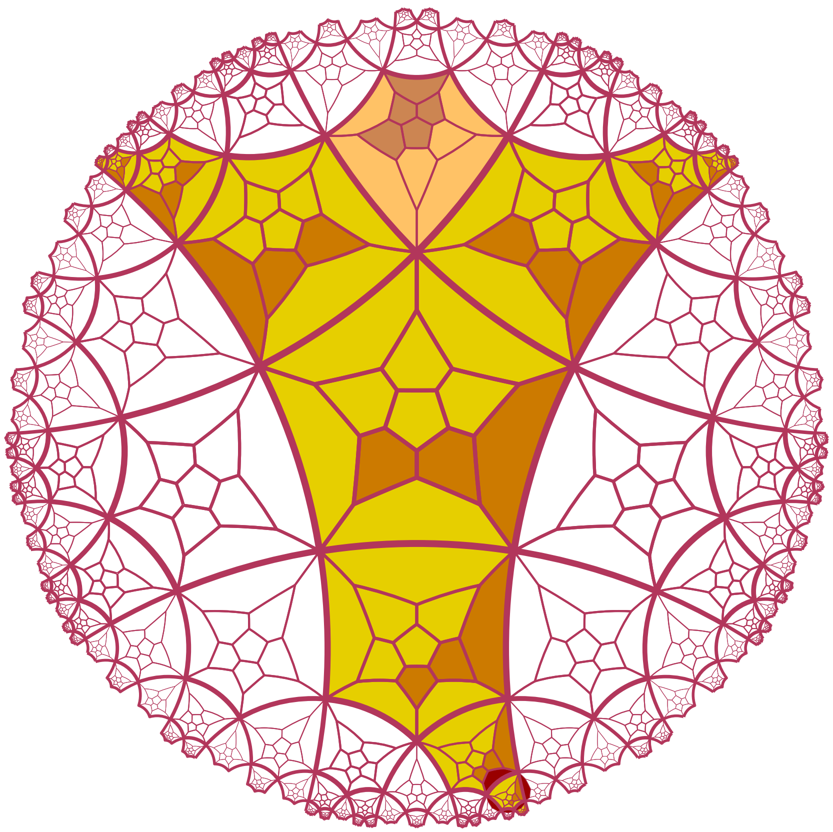


time 6

our implementation:

passive memory switches

motion of the particle:

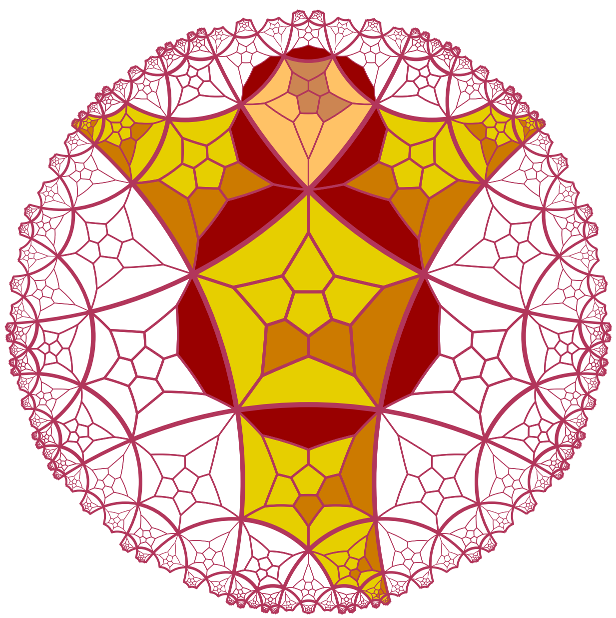


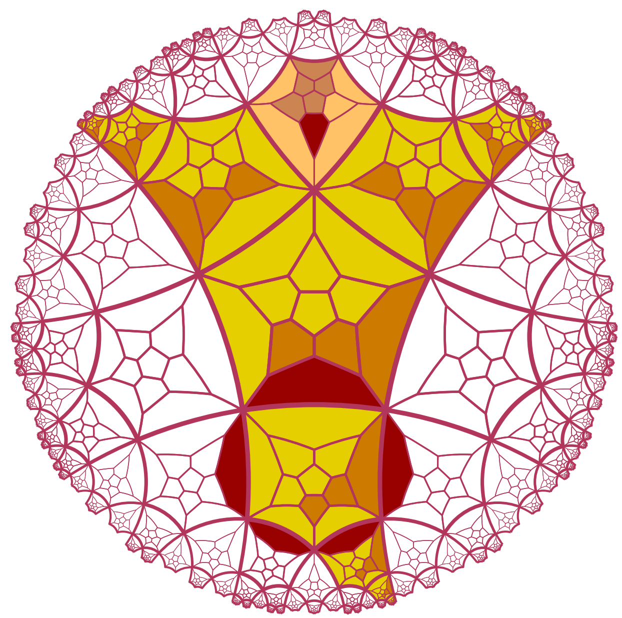
time 7

our implementation:

passive memory switches

note the change at times 4 and 5:





change of selection and a signal sent to the active controller

our implementation:

the **signal** from a **passive** memory switch to the **active** one

take two non-secant planes, Π^p and Π^a

the passive switch on Π_p,

the active one on Π^a,

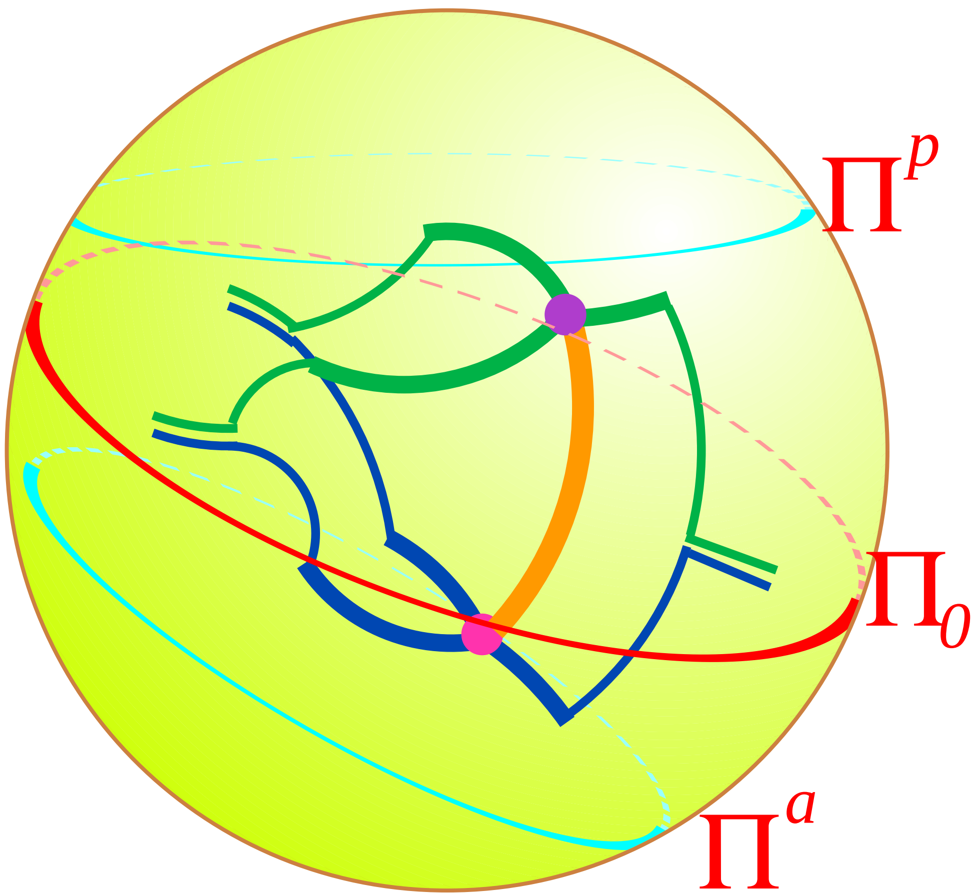
the switches facing each other

through Π_0, the plane of reflection of Π^p onto Π^a

our implementation:

the **signal** from a **passive** memory switch to the **active** one

approximative illustration



our implementation:

the **signal** from a **passive** memory switch to the **active** one

the tiling forces a more complex path from the passive controller to the active one

see *arXiv* paper for more details:

http://arxiv.org/abs/1005.4826

conclusion

and so we proved:

theorem *there is a weakly universal cellular automaton on the dodecagrid with two states involving a truly spacial structure*

conclusion

this result establishes the boundary between decidabilty and weak universality for cellular automata in the $3D$ hyperbolic space

we remain with two questions:

what can be said for strong universality?

what can be said for the hyperbolic plane?

conclusion

best result the hyperbolic plane for weak universality:

a CA in the heptagrid with four states (MM - 2009)

2 states also possible but with a linear structure (MM - 2009)

for strong universality, best result in the hyperbolic plane:

a CA with 9 states, but a linear structure (MM - 2010)

**Thank you
for your attention!**