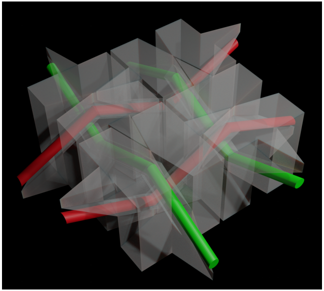
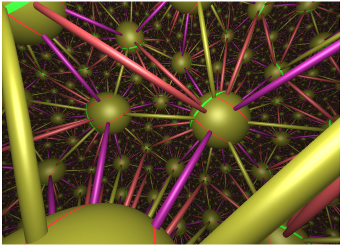
Raytracing in hyperbolic 3-manifolds and link complements

Matthias Goerner

November 13th, 2019

Outline





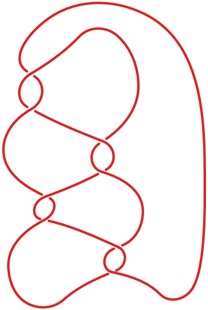
1. Revisit triangulating a link complement.
2. Inside view of a hyperbolic 3-manifold.

Aim: Explicit embedding of hyperbolic triangulation into from link diagram.

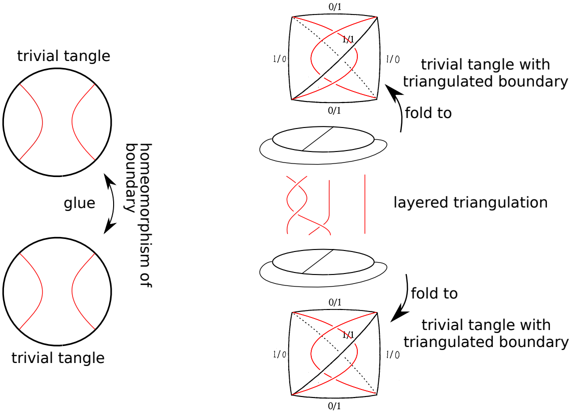
Triangulating a link complement

1. Warm-up: two bridge link complement (ideal).
2. Generic link complement (ideal and finite vertices).
3. Cases where this triangulation 2 admits a hyperbolic structure.
4. Simplification/removing finite vertices.

An example two bridge knot

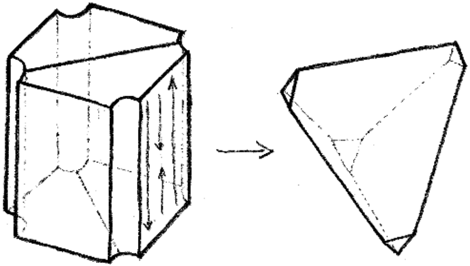


Sakuma-Weeks triangulation for two bridge link

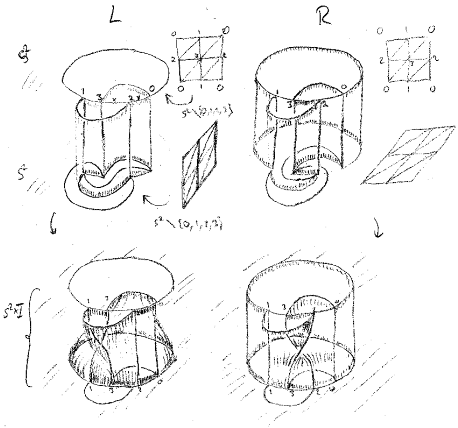


Cubes with diagonals

Easier to visualize: use cubes with diagonals (become tetrahedra of layered triangulation when crushing vertical faces).



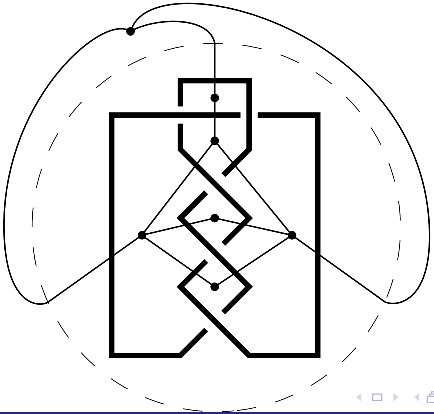
Two bridge links



http://unhyperbolic.org/icerm/

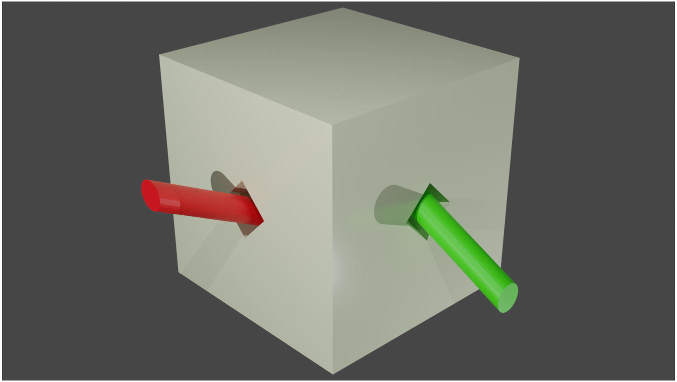
Link diagram

Dual to link diagram: 2-complex of topological squares, each containing exactly one crossing.



Crossing in a box

Replace each topological square by box tangle.



Pinch box

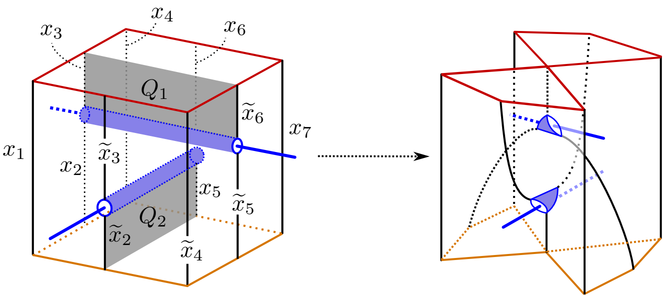
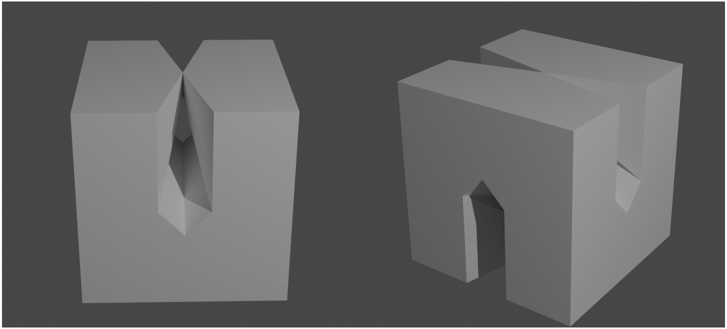


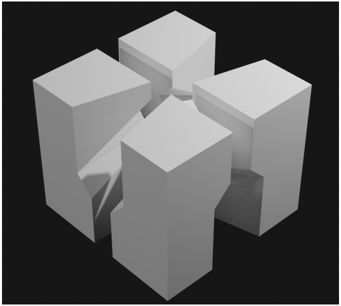
Figure 2: A pinched block

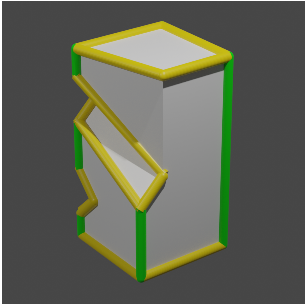
Source: Cho, Yoon, Zickert, *On the Hikami-Inoue conjecture*.

Pinched box



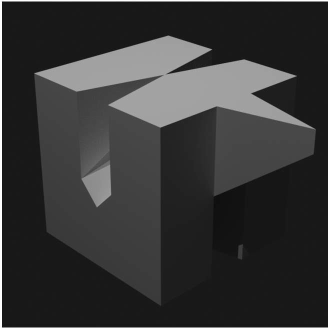
Pinched box can be split into four tetrahedra



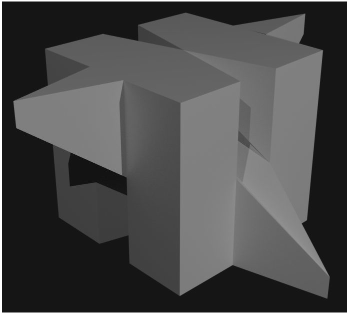


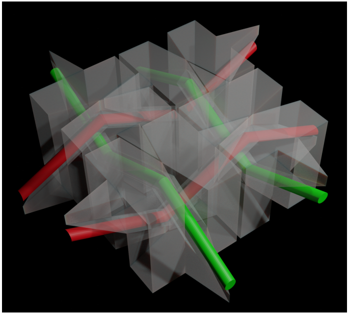
Isotoped neighbors

Isotope neighbors to fill gap from pinching.



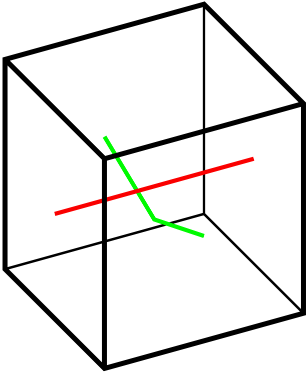
Piece for alternating link



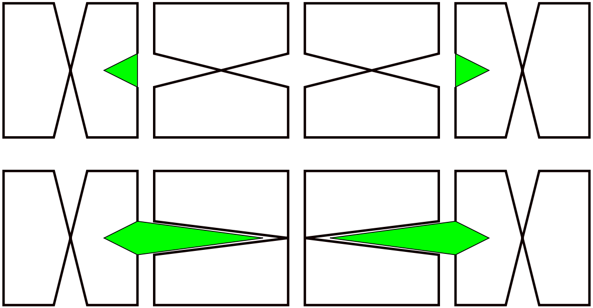


Isotopy for non-alternating links

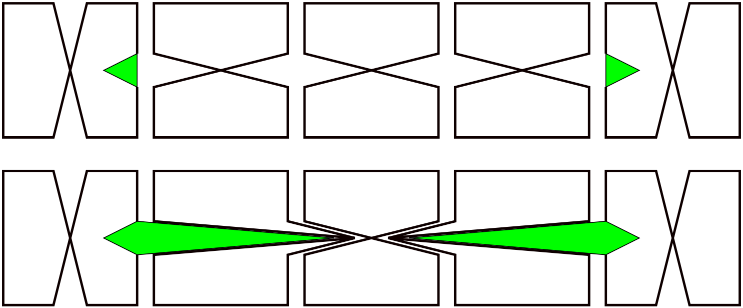
Temporarily straighten segment of link.



Isotopy for non-alternating links



Isotopy for non-alternating links



Geometric structure without removing finite vertices

For the following 23 knots, Orb was able to find a geometric structure on the triangulation without the finite vertices removed:

K4a1	K10a89	K11n157	K12a868
K8a12	K11a266	K11n178	K12a875
K8a15	K11a269	K12a1019	K12a888
K9a29	K11a288	K12a1152	K12n837
K9a37	K11a302	K12a1188	K12n877
K10a121	K11a350	K12a1251	

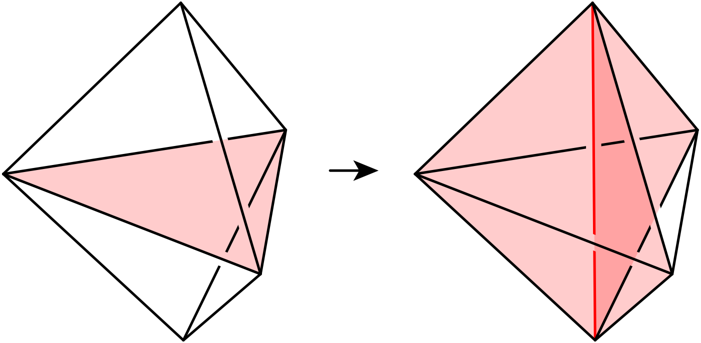
Simplification of triangulation

SnapPy simplifies/removes finite vertices by:

1. Performing 2-3/3-2 moves.
2. 2-0 move (fold two tetrahedra about an edge of order 2).
3. Ungluing a face and gluing in a “triangular pillow with tunnel”.

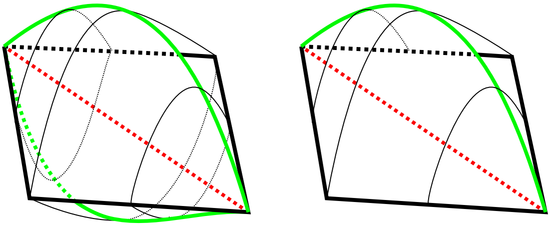
2-3 move

PL-homeomorphism between triangulations straightforward.



2-0 move

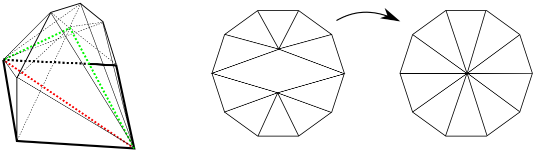
The 2-0 move removes the red order-2 edge and identifies the two green edges and the faces spanned by the green and black edges (pairwise).



From now: use symmetry and only look at one half.

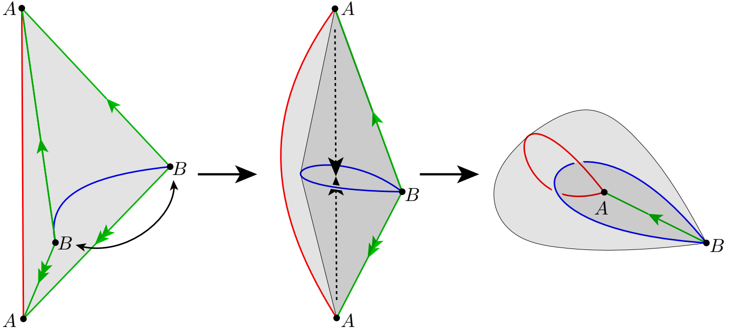
2-0 move

Need to consider a neighborhood of the faces that get identified.



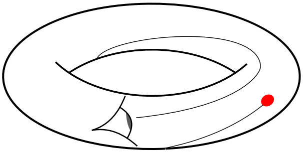
Thanks to Henry Segerman and Saul Schleimer.

Gluing in a "triangular pillow with tunnel"

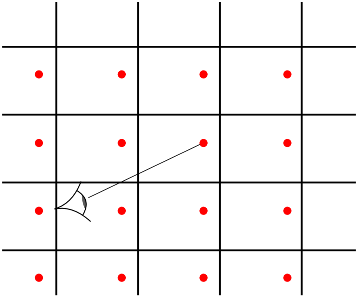


Note: Figure shows one tetrahedron, SnapPy uses two.
Source: Rubinstein, Segerman, Tillman, *Traversing Three-Manifold Triangulations and Spines*.

Technique 1: Draw (rasterize) universal cover



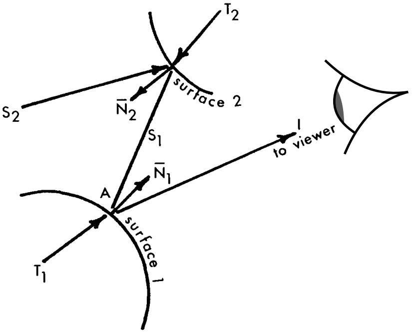
Inside View



Universal cover

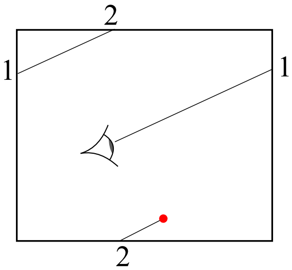
I implemented this using (fixed-function pipeline) OpenGL in 2000 for regular tessellations.

Technique 2: Raytracing



Turner Whitted, *An Improved Illumination Model for Shaded Display*, 1979.

Technique 2: Raytracing



Implemented as GLSL shader in OpenGL 3.2 for SnapPy.

Inside view of a hyperbolic 3-manifold

Available in one of the next versions of SnapPy:

```
M = Manifold("m015")
# Might change to .fly()
M.inside_view() # For triangulation

M = Manifold("m003(-3,1)")
d = M.dirichlet_domain()
d.inside_view() # For Dirichlet domain
```

Thanks to: Henry Segerman et al for initial shader.
Marc Culler for modern OpenGL support on Mac and Linux.

Technique 1 still has applications

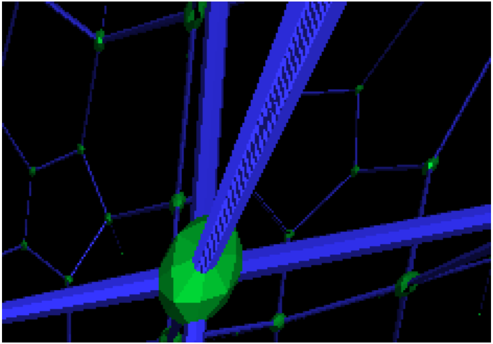
Applications for illustration:

1. Prepare objects (such as geodesic) for raytracing.
2. 2d picture or 3d prints of tessellation by fundamental domains.

Applications for hyperbolic 3-manifolds:

1. Compute length spectrum.
2. Compute maximal cusp area matrix (a_{ij}): neighborhoods of cusp i and j are disjoint if and only if the product of their areas $\leq a_{ij}$ (in writing, Goerner).

Technique 1: Bugs



Double drawing in my first OpenGL implementation: z-Fighting.

Technique 1: Challenges

Challenges:

1. Enumerate each tile only once.
   Easiest: Check whether current tile is ε-close to any previous tile using some tree/hash table structure.
2. Determine when enough tiles have been found.
   Easiest: Use some cut-off size/distance.

This is what Curtis McMullen's *lim* is doing.

Technique 1: Implementations

Challenges:

1. Enumerate each tile only once.
   Elegant: Finite state machine, e.g., Jeremy Kahn's *Circle Limits* (akin to word acceptor of automatic structure).
2. Determine when enough tiles have been found.
   Easiest: Use cut-off distance.
   Note: This is correct if using Dirichlet domain (used by, e.g., SnapPea kernel for length spectrum).

Technique 1: Verified Implementation

Challenges:

1. Enumerate each tile only once.
   Easiest: Check whether current tile is ε-close to any previous tile using some tree/hash table structure.
   Verified: Let ε be radius of a ball contained in fundamental domain. Use interval red-black tree.
2. Determine when enough tiles have been found.
   Verified (without Dirichlet domain): Ensure all external/unglued faces outside of ball to be tessellated.

Goerner, Haraway, Hoffman, Trnkova, *Verified length spectrum* (in progress).