Visualization of Hyperbolic 3-Manifolds

JIMMY HARTFORD

Abstract

In this paper, we use the programming language, Python, along with the software package, SnapPy, to generate fundamental domains of three dimensional hyperbolic manifolds. We also discuss different strategies and tools for visualization and how these are affected when viewed through the Poincare model versus the Klein model

Contents

1 Introduction

In mathematics, one of the primary ways to understand a concept is through visualization. This is especially true with geometry and topology where spacial knowledge of an object is of particular importance. However, visualization can be a difficult concept to master when exploring objects that do not conform to the familiar Euclidean model. The interior of a hyperbolic 3-manifold is an example of a non-Euclidean space. Yet a visual representation of these hyperbolic 3-manifolds is a convenient way to understand its structure.

In this paper, we outline the steps to construct such a visualization of the interior hyperbolic 3-manifolds. This process used information gathered by the program SnapPy.

SnapPy is a program for studying the geometry of 3-manifolds. The current iteration of SnapPy was developed primarily by Marc Culler and Nathan Dunfield and is based on the original SnapPea kernel written by Jeff Weeks.[2]

Working with SnapPy offers several advantages. First, at its heart SnapPy is a Python command-line shell over the original SnapPea. Python is a cross-platform programming language that has a wide use in mathematics and industry. As a result, work done in, or in connection to SnapPy can be easily ported to other applications.

SnapPy also provides a large library of manifolds and essential information about their shapes. In particular, it provides the triangulation of the manifold and shapes of the tetrahedra of that the manifold. Now, we can take a known manifold to view and easily assemble the information needed to build the visualization without constructing the manifold from scratch.

A third advantage for working with SnapPy in the development of a visualization of the interior of hyperbolic manifolds is the developers have already produced tools for visualizing of characteristics of the 3-manifolds in their library such as the Dirichlet domain. This tool can be adapted to display the tessellation of fundamental domain without constructing such a visualization.

While using SnapPy and its resources simplifies the visualization process, there are downfalls as well. Modifying the Dirichlet domain tool prevents us from having to create graphics from scratch, but it also requires overriding and redirecting its inherent functions. As of this writing this has not been entirely completed.

The overall process developed in the following phases: First, retrieve the triangulation of the selected manifold's fundamental domain. Next, use the triangulation to explore and map the shape of the fundamental domain and identify the vertices of the tetrahedra that compose the fundamental domain. Then, we draw the edges of the tetrahedra on the screen.

Finally, we move the eye, or camera, to the origin so that properly depict the movement of light along the geodesics within the manifold, giving the viewer a true presentation of the inside of the manifold.

2 Hyperbolic Manifolds

A manifold is a continuous space that, locally, takes on the properties of the familiar Euclidean space. In other words, for any point, P, inside the manifold, there exists a ball surrounding the point that resembles a ball in Euclidean Space.

Manifolds can have a dimension of any positive integer, n, and we called a manifold having n dimensions a n-manifold. Now, the local properties of an n-manifold resemble $\mathbb{R}^n$. While locally an n-manifold might resemble $\mathbb{R}^n$, globally the n-manifold may not maintain the properties of $\mathbb{R}^n$.

To illustrate this, consider a rectangle. If we were to somehow glue each pair of opposite sides together, we would have an object called a torus. Locally (within the area of the rectangle), this torus will retain the original geometry of the Euclidean rectangle. Globally, the situation is different.

Now that the edges of the rectangle are connected, someone travelling along the surface of the rectangle no longer has to stop once reaching the edge. This traveller could continue beyond the edge and continue in the same direction (or any direction) without end. In this way, the surface of torus is like an infinite plane. However, the surface area of the torus is the same as the original rectangle. So once she passes the edge of the rectangle, she will reenter the rectangle at the opposite side of the rectangle. Continuing along in the same direction, the traveller will reach her starting place. If she were to continue, she would see the same scenery from the first leg of her journey.

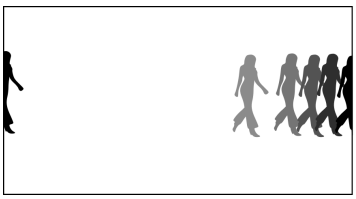

Figure 1: Traveller crossing the edge of the rectangle

From the external point of view we take, observing the torus, it is obvious to us that the traveller has somehow circled back to her starting place on the torus. To anyone living on the surface of the torus, however, they would merely see a plane that continued to infinity in all directions. They would also observe that the world on the surface somehow repeats after moving a certain distance. They might observe that despite the infinite nature of the surface there is only a finite area that is unique.

If our traveller drew a map of her journey, she would draw the original rectangle. Then, she draws another rectangle to show where she crossed the edge. If she continued on a long journey, she would continue adding rectangles to show her process. Similarly, if the traveller moved in any other direction, she could add copies of the rectangle connected to the other edges of the first.

The traveler might observe that copies of the rectangle could be used to tile

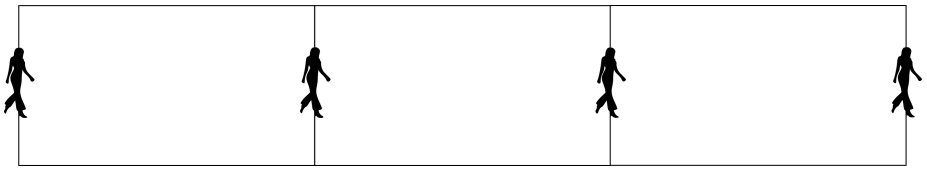

Figure 2: Tiling space with the rectangle

the entire plane. Such a tiling is called the universal cover of the space. Given a space, we say a subset of the space is the fundamental domain if copies of the subset can be used to form the universal cover of the space.

On our 2-manifold, the torus, the fundamental domain is the original rectangle, a polygon. We should note that if we cut open and unfolded the torus along different edges, we would have a different rectangle. This different rectangle could also be used to created the universal cover of the plane.

Now, let us consider a cube. If we somehow glued the opposite faces of the cube together, this would form a 3-manifold. Locally, the 3-manifold looks like the familiar three dimensional Euclidean space, but globally it takes on the properties similar to the torus. Like the 2-manifold, our new 3-manifold has a fundamental domain, the cube that we began with.

Our focus in this paper going forward will be so called hyperbolic 3-manifolds. A hyperbolic 3-manifold is a 3-manifold where the local the geometry is that of $\mathbb{H}^3$.

Much of the work regarding these hyperbolic 3-manifolds is built on discoveries by William Thurston. First, Thurston showed that any knotted link is either, torus, satellite, or hyperbolic. He also showed that if a tube in the form one of these hyperbolic knots is drilled out from a three-sphere, the resulting manifold will have a hyperbolic geometry. [6] The figure-eight knot is an example of such a hyperbolic knot. And we will call the manifold, created by drilling out the this knot, the figure-eight knot complement. This naming reminds us that the manifold is not the knot but what is left once the knot has been removed. Throughout the remainder of the paper, we will use the figure-eight knot complement as our example.

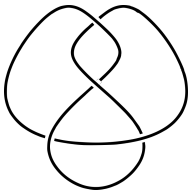

Figure 3: Figure-Eight Knot.

Like the manifold formed by the gluing the faces of the cube has a fundamental domain that is a polyhedron (the cube), the figure-eight knot complement

will as well. The first step in visualising inside of the hyperbolic manifold will be to find the shape of this fundamental domain. But first, let us consider what the inside looks like.

Inside the hyperbolic manifold, the first observation should be that we will observe that the manifold exhibits a hyperbolic geometry. This means that we will need to visualize our model using either a three dimensional Poincare disk or Klein disk to model our manifold (i.e. the interior of a sphere). In the Klein model geodesics are viewed as straight lines, but in the Poincare model, geodesics are viewed as arcs bending towards the origin.

We are interested in what we would see inside the manifold. Let us first assume light rays travel along geodesics. If we place a camera at the center of either the Klein ball or the Poincare ball, we see the same thing, as if we were actually looking along hyperbolic geodesics.

Typically, we view these models from the outside and there is an obvious distinction between the two; however, if we are standing at the center of the model, the bends of the Poincare lines will point directly at us. So, once we are visualizing the the interior of the manifold, there will be no visible difference between the two. Since it will be far simpler later on to draw straight lines in Klein model, we will use the model for any calculations such as distance that might be necessary.

Another observation inside the manifold should be that if we can see far enough in front of us, we will see a copy of ourselves in the distance. And if there is an object near us, we will see that same object, again, in the distance. The same would be true if we looked off in any direction inside the manifold. If our vision is particularly good, we might even be able to see multiple iterations of ourself standing in the distance. This means when we build our visualization of the our manifold, building one fundamental domain will not be sufficient; we will need to make many copies.

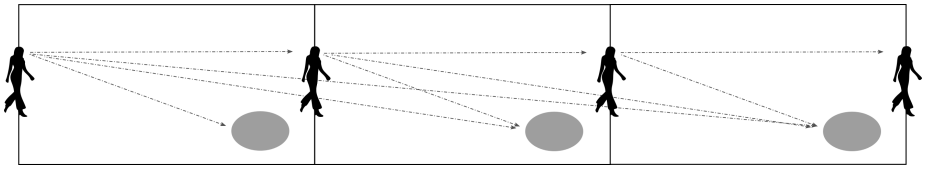

Figure 4: Seeing one's self in the distance.

We need to consider one more observation before moving on. Recall our manifold was formed by drilling out a knot. Where will this knot be in relation to us at the origin? The answer can be a little hard to see but boundary of the manifold will be the edge of the knotted tube as it cuts through our space. As a result, there should be some piece of the fundamental domain that does not glue to some other point. And these pieces will be at infinity. Ultimately, this means that our fundamental domain will be an ideal polyhedron with vertices at infinity. The consequences of this are that all our vertices will be the on boundary of the model (i.e. the surface of the sphere). [7]

3 Triangulation and SnapPy

The first step, in building our visualization is develop a fundamental domain of the manifold. Once, we have created one, we can use it as template for making copies of it. Recall that the manifold really only consists of the one fundamental domain. The extra copies are simply the visualization of what it look like beyond its edge.

Because it is a polyhedron of some kind, the fundamental domain, and really the entire manifold, can be broken into a set of tetrahedra with a list of how they are glued together. This allows SnapPy to store the entire manifold in terms of this triangulation. Figure 5 shows how SnapPy displays the triangulation for the figure-eight knot compliment.

```
([1, 1, 1, 1], [[0, 1, 3, 2], [1, 2, 3, 0], [2, 3, 1, 0], [2, 1, 0, 3]])
([0, 0, 0, 0], [[0, 1, 3, 2], [3, 2, 0, 1], [3, 0, 1, 2], [2, 1, 0, 3]])
```

Figure 5: SnapPy Triangulation for Figure-Eight Knot Complement.

In SnapPy, the tetrahedra are indexed using the convention $0, 1, 2, \ldots, n-1$, where n is the number of tetrahedra in the triangulation. Then each of the four vertices of a given tetrahedron will be given an index, $0, 1, 2, 3$.

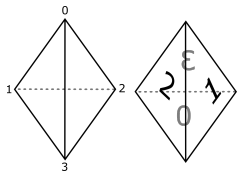

Figure 6: Left: A Tetrahedron with Vertices labeled. Right: A Tetrahredon with Respective Faces labels.

SnapPy also adopts the convention of identifying a given face on the tetrahedron by the vertex opposite to it. For example, $\text{Face}(0)$ will be the face opposite to $\text{Vertex}(0)$.

SnapPy's triangulation provides a list of tetrahedra as well as how each is of the faces in tetrahedra glued to the others. In the table, each row represents the information of the respective tetrahedron with the first row being $\text{Row}(0)$. Then, each row is broken up into two main parts.

The first part is a set of four numbers. This states what tetrahedron each Face on the the first tetrahedron is neighboring. For instance, in the triangulation of the figure-eight knot compliment, Figure 5, the first part of the first row has [1 1 1 1]. The first number "1" says that $\text{Face}(0)$ of $\text{Tetrahedron}(0)$ glues to a face on $\text{Tetrahedron}(1)$. Observe, since there is only one other tetrahedron in our triangulation, all four faces glue to somewhere on $\text{Tetrahedron}(1)$. And

all four faces of Tetrahedron(1) glue back to some face on Tetrahedron(0).

The second part of the row is a set of four sets each, itself, comprised of four elements. This states how each vertex lines up to another. Each subset of the second part corresponds to a face: the first subset corresponds to Face(0) of Tetrahedron(0), the second subset corresponds to Face(1) of Tetrahedron(0), and so on. The element of each subset identifies the respective neighboring vertex in on neighboring tetrahedron identified in the first part. For example, part two of row one has [0 1 3 2], [1 2 3 0], [2 3 1 0], [2 1 0 3]. This tells us that on Face(0) of Tetrahedron(0), verticies $0, 1, 2, 3$ glue to vertices $0, 1, 3, 2$ of Tetrahedron(1), respectively.

We don't yet have a geometric structure for these tetrahedron; the triangulation is just a list vertices and faces that some how glue together. However, the diagrams in Figure 7 help visualize how the two tetrahedra that make the figure-eight knot complement fit together. The diagram on the left shows Tetrahedron(0) and how each Face glues to the Faces of Tetrahedron(1). And the diagram on the right does the same for Tetrahedron(1). The list of numbers in brackets are taken straight from the SnapPy triangulation for the respective Face.

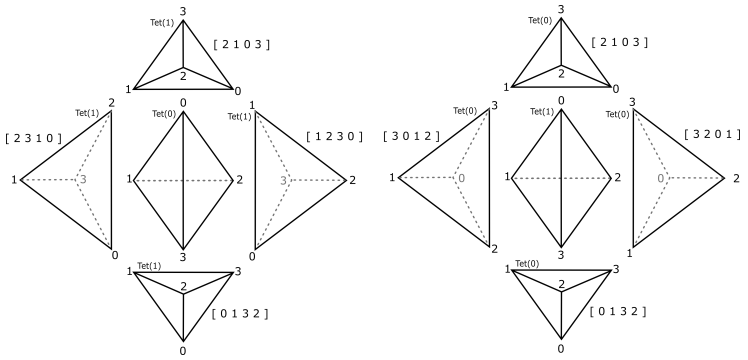

Figure 7: Gluing Diagram for the Figure-Eight Knot Complement.

If we look at Face(3) in Tetrahedron(1), we have the list [2 1 0 3]. Looking at Figure 6, tells us that gluing works in both directions. And later, we will define a transformation that will send one face to the other. This shows that the inverse transformation will send us back to the identity as expected.

```
[0.50000000000 + 0.86602540378*I, 0.50000000000 + 0.86602540378*I]
```

Figure 8: SnapPy Tetrahedron Shapes for Figure-Eight Knot Complement.

The other piece of information we need to build our fundamental domain that SnapPy provides is the tetrahedron shapes. The tetrahedron shapes are a list of complex numbers, one for each tetrahedron.

Recall, the vertices of our fundamental domain (and now the tetrahedra that compose it), will be on the surface of the sphere. As a result we can display the vertex positions as complex projective coordinates.

In the complex plane, positions will be depicted as a pair of complex numbers, $\Big(\alpha, \beta\Big)$, where $\Big(\alpha, \beta\Big) = \lambda\Big(\alpha, \beta\Big) = \Big(\lambda(\alpha), \lambda(\beta)\Big)$, $\alpha, \beta, \lambda \in \mathbb{C}$, and α and β are not both 0. This pair of complex numbers generate a line that will pass through the sphere at a given point. Note, $\Big(1, 0\Big)$ represents the north pole of the sphere and $\Big(0, 1\Big)$ represents the south pole.

Thurston's equations are in terms of these projective coordinates and the solutions are cataloged in SnapPy as the tetraherdon shapes.[3][2]

Using a method developed by Luo [5], we can to find the fourth position of the tetrahedron based on the shape listed in the SnapPy's tetrahedron shapes and positions of the base face we already established:

> Let $\Big(a_1, a_2\Big)$, $\Big(b_1, b_2\Big)$, $\Big(c_1, c_2\Big)$, be pairs of complex coordinates representing the base of our new tetrahedron. Let z be the complex number from SnapPy's list of tetrahedron shapes corresponding to the tetrahedron we are building.
>
> Let $d = a_1 b_2 - b_1 a_2$.
>
> Let $X = \begin{bmatrix} b_2/d & -b_1/d \\ -a_2/d & a_1/d \end{bmatrix}$, and $\begin{bmatrix} c_1' \\ c_2' \end{bmatrix} = X \begin{bmatrix} c_1 \\ c_2 \end{bmatrix}$.
>
> Our new position, $\Big(d_1, d_2\Big) = X^{-1} \begin{bmatrix} -z/c_1' \\ -1/c_2' \end{bmatrix}$.

Once we have built one tetrahedron in the the triangulation, we can use it as the starting point for the next.

After we find all the vertices of the fundamental domain, we must convert those coordinates into $\mathbb{R}^3$ so that our graphics program can draw them.

4 Fundamental Domain

The triangulation in SnapPy depicts the manifold combinatorically. From this we want to build a fundamental domain so we have a geoemetric structure that we can visualize. To build the fundamental domain, we must systematically look at each tetrahedron in the triangulation, and find its vertices. Then, we look at its neighbors.

To do this, we must pick a starting point. And systematically explore every face of every tetrahedron in the triangulation. The process described here is based on Python code written by Henry Segerman, a version of which is available in Appendix A.

Here is the triangulation for the figure-eight knot complement again for reference as we explore its tetrahedra.

```
([1, 1, 1, 1], [[0, 1, 3, 2], [1, 2, 3, 0], [2, 3, 1, 0], [2, 1, 0, 3]])
([0, 0, 0, 0], [[0, 1, 3, 2], [3, 2, 0, 1], [3, 0, 1, 2], [2, 1, 0, 3]])
```

Let n be the number of tetrahedra in the triangulation.

Let $L = \{0, 1, \ldots, n-1\}$. We will use L to determine if we have looked inside a particular tetrahedron in the triangulation. Then, we will look at a neighboring tetrahedron with index, i. If $i \in L$, then we will move to that tetrahedron, remove i from the set L, and explore Tetrahedron$\big(i\big)$. If $i \notin L$, then we have already look in there and we can move on.

Let us define $n \times 4$ arrays, P, V, where rows represent the corresponding tetrahedron and positions in each row correspond to the respective vertex of the corresponding tetrahedron.

Let the entries of P be one of the following: "i" for interior, "e" for exterior, or "-" representing a face we have not traversed. Let us set all entries in P to "-". This array will provide several pieces of information. First, during the process an "-" entry will tell us that this is a face we need to look at. Once the process is completed there will be only "e" and "i" entries. This will tell us if that face is an interior face of exterior face for the fundamental domain we have built.

In the triangulation, each tetrahedron has vertices indexed $0, 1, 2, 3$. But we want also each vertex of the fundamental domain to have a unique index so we can assign a unique position that we can visualize.

As we go along we will indicate a vertex in a given tetrahedron by the tetredhon index followed by its local face index, i.e. "Tetrahedron$\big(i\big)$: Face$\big(j\big)$". When we want identify a vertex by it global index, we will say Vertex$\big(x\big)$.

Let the entries of V be unique integers representing the global index for the Vertex$\big(i\big)$ of Tetrahedron$\big(j\big)$ corresponding to entry $v_{i,j} \in V$.

Each time we discover an unexplored tetrahedron in the tetrahedron, we will also discover a new vertex whose coordinates we need to find. With the positions representing the face that we passed through to discover the new tetrahedron being used as the base, we will use the method method in the previous section to establish this new position. we will start with the first three positions begin $\Big(1, 0\Big)$, $\Big(0, 1\Big)$,$\Big(1, 1\Big)$, representing the top, bottom, and a point on the equator of the sphere.

Since, we can choose any tetrahedron to start at, it makes sense to start exploring from Tetrahedron$\big(0\big)$ in the figure-eight knot complement.

So, let, $i = 0$. First, we check whether we have been here before. This is the first one; so, it is new. Now, we remove i from the list L.

Since this is the first tetrahedron in our triangulation, we know that its local vertex indices will be the same as it local positions. So, row 0 of V will be [0 1 2 3]. This indicates that Tetrahedron$\big(0\big)$: Face$\big(0\big)$ is indexed globally by Vertex$\big(0\big)$ and Face$\big(1\big)$ is Vertex$\big(1\big)$ Face$\big(2\big)$ is Vertex$\big(2\big)$ and Face$\big(3\big)$ is Vertex$\big(3\big)$.

Tetrahedron$\big(0\big)$ has four faces. Again it makes sense to start with Face$\big(0\big)$. Define j_i be the local face index for our current tetrahedron. And let $j_0 = 0$.

$P_{0,0}$ is "-". So, we know this is an unexplored face. Looking back at the triangulation table, we see that on Tetrahedron$\big(0\big)$: Face$\big(0\big)$ glues to Tetrahedron$\big(1\big)$: Face$\big(0\big)$.

We set $P_{0,0}$ = "i" and $P_{1,0}$ = "i".

The vertices that make up Tetrahedron$\big(0\big)$, Face$\big(0\big)$ are the other three vertices, so 1,2,3. And they glue to to faces on Tetrahedron$\big(1\big)$ 1, 3, 2 (by the triangulation table above). This tells us that Tetrahedron$\big(1\big)$: Face$\big(1\big)$ is globally the same vertex as Tetrahedron$\big(0\big)$: Face$\big(1\big)$. Similarly, Tetrahedron$\big(0\big)$: Face$\big(2\big)$ is Tetrahedron$\big(1\big)$: Face$\big(3\big)$, and Tetrahedron$\big(0\big)$: Face$\big(3\big)$ is Tetrahedron$\big(1\big)$: Face$\big(2\big)$. This leaves Tetrahedron$\big(1\big)$: Face$\big(0\big)$ which will be a new point. The next available integer is 4. So, Tetrahedron$\big(1\big)$: Face$\big(0\big)$ is Vertex$\big(4\big)$. And now, row 1 of V will be [4 1 3 2].

Now, we need to find the position of Face$\big(1\big)$ Tetrahedron$\big(0\big)$. We started with positions $\Big(1,0\Big)$, $\Big(0,1\Big)$,$\Big(1,1\Big)$. So these will be Vertex$\big(0\big)$, Vertex$\big(1\big)$, Vertex$\big(2\big)$, respectively. We use the process described in the previous section and find Vertex$\big(3\big)$ is at $\Big(0.5+\frac{\sqrt{3}}{2}i,1\Big)$.

And now we will the use base vertices Vertex$\big(1\big) = \Big(0,1\Big)$, Vertex$\big(2\big) = \Big(1,1\Big)$, Vertex$\big(3\big) = \Big(0.5+\frac{\sqrt{3}}{2}i,1\Big)$, to find Vertex$\big(4\big)$, the result of which is $\Big((-0.5-\frac{\sqrt{3}}{2}i),(-1.5-\frac{\sqrt{3}}{2}i)\Big)$. We now have built Tetrahedron$\big(1\big)$. Staying here, we will look at its faces.

Let $i = 1$ and remove 1 from L. Let $j_1 = 0$. $P_{1,0}$ = "i". This tell us that we have already looked in this direction (in particular, this where we came from).

Let $j_1 = 1$. $P_{1,1}$ is "-". So, we know this is an unexplored face. Looking back at the triangulation, we see that on Tetrahedron$\big(1\big)$: Face$\big(1\big)$ glues to Tetrahedron$\big(0\big)$: Face$\big(2\big)$.

Now we look at L and see if Tetrahedron$\big(0\big)$ needs to be explored. $0 \notin L$, so Tetrahedron$\big(0\big)$ does not need to be explored. (This means that we have completely mapped Tetrahedron$\big(0\big)$ or we are in the process and got sidetracked looking in our current location. In fact, there is a total of five unique positions in the fundamental domain of the figure-eight knot complement and at this point, we have actually found all five. We could stop now; however, we need to verify that this is the case).

Since, we have already looked here, we need to explore Tetrahedron$\big(0\big)$ we set $P_{1,1}$ = "e" and $P_{0,2}$ = "e". And move on to $j = 2$. (If it were not then this face would not be on the exterior of the face so we would mark it with an "i", Then, we would start the whole process again inside this new tetrahedron. And systematically look at each of its faces).

Now, let $j_1 = 2$. Tetrahedron$\big(1\big)$: Face$\big(2\big)$ glues to Tetrahedron$\big(0\big)$: Face$\big(1\big)$. Again, we know Tetrahedron$\big(0\big)$ has been explored, so we set $P_{1,2}$ = "e" and $P_{0,1}$ = "e". And we will find a similar result at $j_1 = 3$, so we can set $P_{1,3}$ = "e" and $P_{0,3}$ = "e".

We have now explored every face of Tetrahedron$\big(1\big)$. Next, we move back to where we left off in Tetrahedron$\big(0\big)$.

Let $j_0 = 1$, $P_{0,1}$ is not "-", so we can move on.
Let $j_0 = 2$, $P_{0,2}$ is not "-", so we can move on.
Let $j_0 = 3$, $P_{0,3}$ is not "-", so we can move on.
Now, we have explored all of the faces of Tetrahedron$\big(0\big)$.

The next step, will be to set i to the next entry in our list L, of tetrahedra left to do. In this case, we see L is now empty, so we are done.

Note: We chose to start with Tetrahedron$\big(0\big)$ and look at faces in order 0, 1, 2, 3 each time. This is a somewhat arbitrary choice. We could instead start with the last tetrahedron and move in the opposite direction. The same could have been done for faces. Or we could even find a random order (assuming we had way to track that each face of each tetrahedron was explored.) The result of this would be a different polyhedron for the fundamental domain. However, once we begin building and drawing other copies of the fundamental domain, there should be no detectable difference.

Completing this procedure will assign a unique global index to every vertex in the fundamental domain, identify the coordinates of each of these unique vertices, and tells us which Faces are interior. The list of interior versus exterior Faces will be needed in the following section.

For the figure-eight knot complement we have

Face Types:
[i e e e]
[i e e e]

Global Vertex Indicies:
[0 1 2 3]
[4 1 3 2]

Vertex Coordinates:
0: $\Big(1, 0\Big)$, 1: $\Big(0, 1\Big)$, 2: $\Big(1, 1\Big)$,
3: $\Big(0.5 + \frac{\sqrt{3}}{2}i, 1\Big)$, 4: $\Big((-0.5 - \frac{\sqrt{3}}{2}i), (-1.5 - \frac{\sqrt{3}}{2}i)\Big)$

5 Iterations of the Fundamental Domain

Now, we that we have built a fundamental domain. We need to make copies at different locations in the Klein model. To do this, we need to create a group of matrices that will map the location of one face to another.

Again, using the figure-eight knot as an example, suppose we wanted to find the fundamental domain that is glued to Tetrahedron$\big(0\big)$: Face$\big(3\big)$. First, we need to identify its coordinates. Tetrahedron$\big(0\big)$: Face$\big(1\big)$ is composed of Tetrahedron$\big(0\big)$: Face$\big(0\big)$, Face$\big(1\big)$, and Face$\big(2\big)$ which have global indices, Vertex$\big(0\big)$, Vertex$\big(1\big)$, and Vertex$\big(2\big)$, respectively.

Using the SnapPy triangulation table, we see that these vertices glue to Tetrahedron$\big(1\big)$: Face$\big(2\big)$, Face$\big(1\big)$, and Face$\big(0\big)$, respectively. This gives us Vertex$\big(3\big)$, Vertex$\big(1\big)$, and Vertex$\big(4\big)$.

Since the new copy of the fundamental domain will be such that it is built on the coordinates of the previous copy, we need a map that will send the starting coordinates of the new fundamental domain to finishing coordinates of previous.

To do this we will need to find a matrix M that send $\Big(1,0\Big)$, $\Big(0,1\Big)$, $\Big(1,1\Big)$ to each set of starting positions, and a matrix N that will sends $\Big((-0.5 - \frac{\sqrt{3}}{2}i), (-1.5 - \frac{\sqrt{3}}{2}i)\Big)$, $\Big(0,1\Big)$, $\Big(0.5 + \frac{\sqrt{3}}{2}i, 1\Big)$ to each set of starting positions. Then by multiplying M and N^{-1} together, we have a will have map that sends the ending coordinates to the new starting coordinates.

Let (p_1, p_2), (q_1, q_2), (r_1, r_2) be the triple of complex coordinates for our face we want to map to.

Let $X = \begin{bmatrix} p_1 & q_1 \\ p_2 & q_2 \end{bmatrix}^{-1}$, and Let $\begin{bmatrix} y_1 \\ y_2 \end{bmatrix} = X \begin{bmatrix} r_1 \\ r_2 \end{bmatrix}$.

And $M = \begin{bmatrix} p_1 y_1 & q_1 y_2 \\ p_2 y_1 & q_2 y_2 \end{bmatrix}$.

This will yield:

$M = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$, and $N = \begin{bmatrix} -.5 - \frac{\sqrt{3}}{2}i & 0 \\ -1.5 - \frac{\sqrt{3}}{2}i & -.5\frac{\sqrt{3}}{2}i \end{bmatrix}$.

Now, our new transform, $A = M \times N^{-1} = \begin{bmatrix} -.5 - \frac{\sqrt{3}}{2}i & 0 \\ -1 & -.5\frac{\sqrt{3}}{2}i \end{bmatrix}$.

Note that if we map the Tetrahedron$\big(0\big)$: Face$\big(0\big)$ to Tetrahedron$\big(1\big)$: Face$\big(0\big)$, neighboring internal faces, we would get the identity matrix. This makes sense because the coordinates of one face are the same as the coordinates of the other. This will be true for all interior faces. As a result we can immediately ignore these faces. It should also be clear that there will be an even number of interior faces; for each interior face in a given tetrahedron, there exists an other interior face other some of tetrahedron to which it is glued.

After ruling out the interior faces of the figure-eight knot complement, we are left with six exterior faces. And for each face, we have a matrix transformation. Note there will be an even number number of exterior faces as well. For every

gluing, there is inverse gluing that creates a fundamental domain in the other direction. So, for a transformation matrix A, there is another transformation matrix a such that $a = A^{-1}$.

Suppose $v = [a, b]$ is the position for a vertex within the original fundamental domain. Then vA will be the corresponding position of vertex v in the copied fundamental domain generated by matrix A.

Adding the identity, I, we have a group, $G = \langle I, A, B, C \rangle$ which will act on the coordinates of the fundamental domain vertices. In particular, it is an infinite group which will allow us to tile the entire manifold with copies of the fundamental domain.

Up until this point we have been using complex projective coordinates. This was convenient when we were trying fix positions on the surface of the sphere. Now, we beginning to look what the interior of the sphere looks like. As a result, we need to map our projective coordinates to the Klein disk model.

Suppose we have a pair of complex numbers, $(X, Y) = (x_0 + ix_1, y_0 + iy_1)$.

Let $d = x_0^2 + x_1^2 + y_0^2 + y_1^2$. Then,
$x = 2(x_0(y_0) + x_1(y_1))/d.$
$y = 2(y_0(x_1) - x_0(y_1))/d.$
$z = (x_0^2 - y_0^2 + x_1^2 + y_1^2)/d.$

For instance, $(1, 0) \longmapsto (0, 0, 1)$ and $(0, 1) \longmapsto (0, 0, -1)$

Having coordinates in three dimensional space will allow us draw edges and faces within the sphere. Before we are ready for this, there is one last matter to address. Even though our G will generate an infinite number of copies of the fundamental domain, if we are generating a viewable model of the manifold, we only need to build a finite number of copies that will be visible to the viewer. Therefore, we only need to generate copies that will be within a predetermined range, r, from the original.

Because it is hard to know ahead of time what the actual polyhedral shape for the fundamental domain will be, it is convenient to simply average the coordinates of all the vertices in the fundamental domain and call the resulting position the 'center.' Now, we can compare the distances between the center our original fundamental domain constructed in Section 4 and the center of a copy generated by our group, G, acting on the original points and determine if the that distance is less than r.

Within the Klein model of hyperbolic space, the distance between two points p,$q \in \mathbb{H}^3$ is $\mathrm{d}(p, q) = \frac{1}{2} \log \left(\frac{|aq||pb|}{|ap||qb|} \right)$, where a. b are two points on the sphere and are contained in the line passing through p, q, and aq, bq, ap, bp are Euclidean distance between the each pair of points.

To find a and b we need to find the intersection between the sphere and the line. Suppose $p = (x_p, y_p, z_p)$ and $p = (x_q, y_q, z_q)$. Let $a = (x_a, y_a, z_a)$ and $b = (x_b, y_b, z_b)$ be the intersection with our sphere, $x^2 + y^2 + z^2 = 1$ and our line

parameterized as

$$\begin{aligned} x &= x_p + u(x_q - x_p) \\ y &= y_p + u(y_q - y_p) \\ z &= z_p + u(z_q - z_p) \end{aligned}$$

Substituting these values into our equation for the sphere will give us the quadratic $au^2 + bu + c = 0$ with

$$\begin{aligned} a &= (x_q - x_p)^2 + (y_q - y_p)^2 + (z_q - z_p)^2 \\ b &= 2\left((x_q - x_p)x_p + (y_q - y_p)y_p + (z_q - z_p)z_p\right) \\ c &= x_p^2 + y_p^2 + z_p^2 - 1 \end{aligned}$$

Using the quadratic formula, we can solve for u; and there will of course be two values for u. Plugging u into the line parameterization equations will give (x_a, y_a, z_a) and (x_b, y_b, z_b).[1]

Here we can easily find the distances aq, bq, ap, bp and plug them into our Klein distance formula and compare the value to r. The specific value for r is a bit harder to pin down. Factors for determining r include your computer's memory, graphic card, and the structure of the manifold. If the manifold is overly intricate, additional elements might become more distracting than informative.

6 Visualizing the Manifold

Now we are ready to draw images on the screen. The simplest thing to do is simply feed a list of triangular faces and where they sit in $\mathbb{R}^3$. Doing so will create an image similar to Figure 9. This image provides us with the overall shape and structure of the fundamental domain. We can observe that the five vertices we discovered in Section 4 are present, labelled in order. We can also see that the shape of the fundamental domain consist of two tetrahedra glued together, as we would expect.

While our point of view is still outside the model, we can take a quick peek at how our fundamental domain looks in the Poincare Model (Figure 10). Note how the edges all curve towards the surface of the sphere, i.e. infinity. Recall that the vertices of our tetrahedra are all at infinity. And even though we visualize them here as finite objects, they represent locations that we would not be able to reach if we were inside the manifold.

We should recognize also that in the manifold, these faces don't really exist. In the actual manifold, these faces are simply windows or doorways that someone inside the manifold might pass through.

If we want to remove the faces so that we only draw edges of the faces, to allow us to view the inside of the fundamental domain, we need construct objects in our model that will represent these edges.

There are a variety of ways to this. In fact, we are not limited to drawing the edges. These edges are geodesics in hyperbolic space so they are relevant things

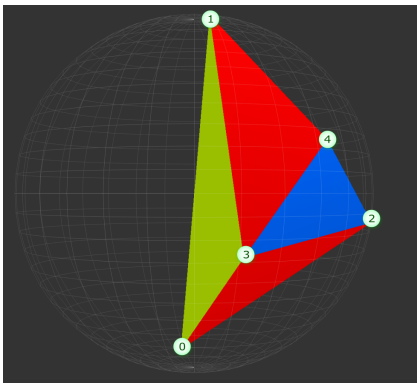

Figure 9: Faces of the Fundamental Domain of the Figure-Eight Knot Complement.

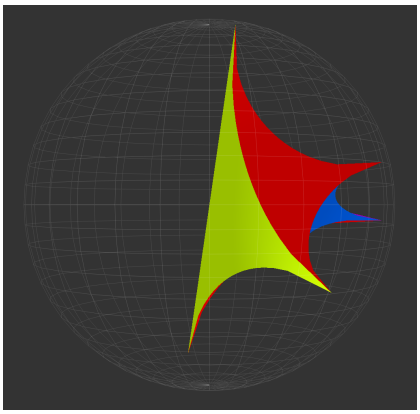

Figure 10: Edges of the Fundamental Domain in the Poincare Model.

to draw because they explictly show the structure of the space we are modeling. However, with the location of the (here in our figure-eight knot complement, five) vertices of the fundamental domain we could draw almost anything as long as it fits inside the polyhedron.

The edges drawn in Figure 11 were drawn with the following method. For any vertex on the tetrahedron, there will be three faces adjacent. Find the centroid of each adjacent face. We, then, use these positions as a weight on our vertex, moving it towards each centroid along the surface of each face away from the true vertex. This will generate three new vertices for each true vertex. Then we draw faces connecting the appropriate weighted vertices, running parallel to the true edge. And then finally draw a triangular face to connect the three weighted vertices generated by the same true vertex.

This gives us visiable edges like Figure 11 that are visiable as we rotate the image.

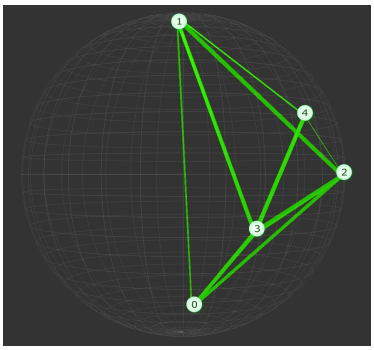

Figure 11: Edges of the Fundamental Domain of the Figure-Eight Knot Complement.

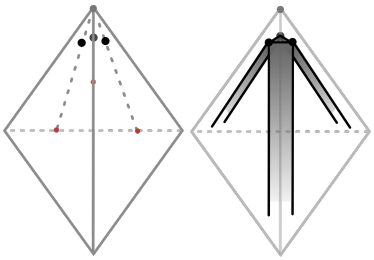

Figure 12: Weighted Vertices and Weighted Edges.

Now that we have had a chance to look at the fundamental domain, let us look at how the image looks when we add the additional copies of the fundamental domain generated in Section 5. Figure 12 displays copies of the fundamental domain in addition to the original one from Figure 11. With all the additional edges drawn, it becomes difficult to see in a static picture. However, each edge drawn in Figure 11 is a part of a tetrahedron which is in turn part of a copy of our fundamental domain. This perspective may give the impression that our manifold contains all of these polyhedrons. This is incorrect. Recall from Section 2, that the torus we made still contained only that one rectangle. This infinite nature of the torus came from the notion that we look (and move) beyond the edge of the rectangle and view what layed behind us on the other side.

This is the case without figure-eight knot complement manifold images. Figures 9-12 show the external view of the manifold. We are three dimensional beings. So we can view the plane from an external view and see how the torus was made up of just that one rectangle. Our traveler, however, could not (without assistance) see how the manifold is formed. The same is true for these hyperbolic manifolds we have been working with throughout this paper.

If we were to enter the manifold and look around just as our traveler did on

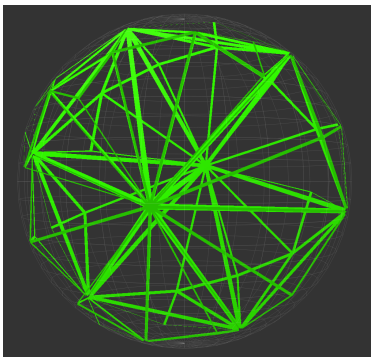

Figure 13: Visualization of Figure-Eight Knot Complement.

the torus, we might see an image like Figure 13. Now, the the additional copies of the fundamental domain are placed in the correct perspective. The smaller shapes in the background represent objects in the distance, beyond the edge of the fundamental domain.

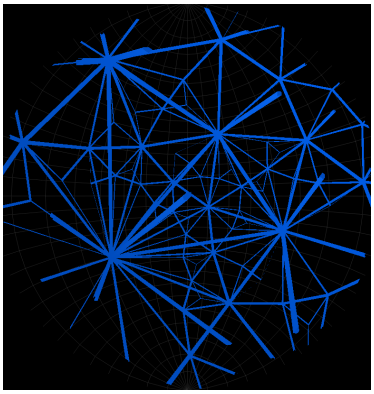

Figure 14: Inside View of the Figure-Eight Knot Complement.

7 Next Steps

The images shown throughout Section 6 were generated using SnapPy's Dirichlet domain tool as discussed in the introduction. The original plan for this project was to modify this tool for the purpose of displaying this interior view, as seen in Figure 13. The goal was also create functions with this tool that would allow someone simulate flying through the manifold so a person could truly explore (and get a sense of) the inside of the hyperbolic manifolds that SnapPy works with. Ultimately the plan is to continue working on this project to complete

this final goal and have a polished program that SnapPy's developer could easily implement as part of a future SnapPy release. In this final section, I will discuss the final steps necessary to complete this goal.

First, the Dirichlet domain tool works on two levels. One is the user interface; the other is the deeper program that works directly with the graphics card. At the user interface level, I have successfully injected commands that simulate some movement inside the manifold. However, this approach is extremely inefficient. The result is a few second lag for each step in a given direction. So this is an impractical approach.

The deeper level code appears to be a better solution, However at this time tracing through its components well enough to override the native functions has not been successful. This has led me to export the sets of vertex positions created each time a move is executed as a OBJ files to a program called Blender which has an animation function. By taking a shot of each each mesh frame by frame, reasonable video can be generated. But this method provides no user interaction.

The method for movement is based on [4]. The effect is similar to the warp drive effect, where the view remains stationary and the universe move around them.

First, we will decide on a velocity vector, $v = (x, y, z)$, that will move the fundamental domain past us by some distance using affine transformations.

Let $M = \begin{bmatrix} 0 & 0 & 0 & x \\ 0 & 0 & 0 & y \\ 0 & 0 & 0 & z \\ x & y & z & 0 \end{bmatrix}$.

We let $c_1 = sinh(||v||^2)$, and $c_2 = cosh(||v||^2) - 1$.

Now, we have a transformation, $T = I_4 + c_1(M) + c_2(M)^2$, where I_4 is the 4x4 identity matrix. After ensuring T is orthogonal, we apply T to every vertex position in the homogeneous coordinates for the Klein model in $\mathbb{R}^4$.

The final step in our moving visualization is to move beyond the edge of our fundamental domain as our traveler did. This too will be a trick of the eye. Since each fundamental domain is identical, we can calculate when we approaching the edge of the fundamental domain and use the transforms from G in Section 5 to move us back to a place at the front of the fundamental domain allowing us to recycle the edges we have already drawn and not generate anything new. This piece is also a work in progress. The original matrices in G were created for use in the projective plane and we need a proper isomorphism from PSL(2, $\mathbb{C}$) to SO(3,1) that we can use along side our affine transformations.

Finally, there is some optimization that is necessary in the creation of our group action, G from Section 5.

A Python Procedure to Build the Fundamental Domain

The following segment of code was used inside Python to construct the fundamental domain of a given manifold generated by SnapPy following the process descibed in Section 4 by passing the develop_fd function the triangulation and tetrahedron shapes from our manifold.

```
def develop_fd(trianglugation, tet_shpes):

    strt_indx = 0
    vert_glug = triangulation
    num_tet   = len(vert_glug)
    tet_to_do = [] # tets we need to explore
    face_type = [] # 'e' or 'i'
                   #identifies whether the face is on the inside or
                   #outside of FundamentalDomain.
    vert_indx = [] # Global indicies for vertices in FundamentalDomain

    # we initialize these variables
    for i in range(num_tet):
        tet_to_do.append(i)
        face_type.append(['-','-','-','-'])
        if i!=0: #do one fewer of these
            vert_indx.append(['-','-','-','-'])
    vert_indx.insert(strt_indx, [0,1,2,3])
    vert_psns = [[1,0],[0,1],[1,1],[tet_shpes[strt_indx],1]]

    # this will walk through the fundamental domain and find all tetrahedrons
    explore_tet(strt_indx, vert_glug, tet_to_do,
                face_type, vert_indx, vert_psns, tet_shpes)

    return face_type, vert_indx, vert_psns

# recursively builds fundamental domain
def explore_tet(crrnt_tet, vert_glug,  tet_to_do,
                face_type, vert_indx, vert_psns, tet_shpes):

    vert_dict = {0:(3,2,1), 1:(2,3,0), 2:(1,0,3), 3:(0,1,2)}

    tet_to_do.remove(crrnt_tet)
    for crrnt_fce in range(4):
        if face_type[crrnt_tet][crrnt_fce] == '-':
            # then we haven't look through this face

            # this is what is through this face
```

```
#crrnt_fce and nghbr_fce are using local indicies
nghbr_tet = vert_glug[crrnt_tet][0][crrnt_fce]
nghbr_fce = vert_glug[crrnt_tet][1][crrnt_fce][crrnt_fce]

if nghbr_tet in tet_to_do:
    #then this is the first time we have traversed to this tet

    face_type[crrnt_tet][crrnt_fce] = 'i' #interior face
    face_type[nghbr_tet][nghbr_fce] = 'i' #interior face

    glued_tet = vert_glug[crrnt_tet][1][crrnt_fce]

    for vert in range(4):
        if vert != crrnt_fce:  #match on vertices that are glued
            vert_indx[nghbr_tet][glued_tet[vert]]
                                = vert_indx[crrnt_tet][vert]

    # This will be a newly identified vertex
    new_vertx = glued_tet[crrnt_fce]
    # This will be new_vertx's global index in vert_indx
    new_index = len(vert_psns)

    vert_indx[nghbr_tet][new_vertx] = new_index

    # We need to identify base vertices so we can locate the
    # position of the new vertex. find_new_position will find
    # it then we append it to vert_psns
    vert_zero = vert_indx[nghbr_tet][vert_dict[new_vertx][0]]
    vert__one = vert_indx[nghbr_tet][vert_dict[new_vertx][1]]
    vert__two = vert_indx[nghbr_tet][vert_dict[new_vertx][2]]

    new__posn = find_new_position(
                        vert_psns[vert_zero],
                        vert_psns[vert__one],
                        vert_psns[vert__two],
                        tet_shpes[nghbr_tet])
    vert_psns.append(new__posn)

    # now, we explore nghbr_tet
    explore_tet(strt_indx, vert_glug, tet_to_do,
                face_type, vert_indx, vert_psns, tet_shpes)
else:
    # we have already been here, and we now that these are
    # exterior faces
    face_type[crrnt_tet][crrnt_fce] = 'e'
    face_type[nghbr_tet][nghbr_fce] = 'e'
```

References

[1] Paul Bourke. Intersection of a line and a sphere (or circle). Available at `http://portal.ku.edu.tr/~cbasdogan/Courses/Robotics/projects/IntersectionLineSphere.pdf`, 1992.

[2] Marc Culler, Nathan M. Dunfield, and Jeffrey R. Weeks. SnapPy, a computer program for studying the topology of 3-manifolds. Available at `http://snappy.computop.org` (11/01/2015).

[3] Stavros Garoufalidis, Matthias Groerner, and Christian K. Zickert. Gluing equations for PGL(n, $\mathbb{C}$) - representations of 3-manifolds. 2015.

[4] Vi Hart and Henry Segerman. Available at `https://github.com/vihart/webVR-playing-with/tree/master/js`.

[5] Feng Luo, Stephan Tillmann, and Tian Yang. Thurstons's spinning construction and solutions to the hyperbolic gluing equations for closed hyperbolic 3-manifolds. *Procedings of the American Mathematical Society*, 141(1), 2012.

[6] Jessica Purcell. Hyperbolic knot theory. Available at `math.byu.edu/~jpurcell/papers/hyp-knot-theory.pdf`.

[7] Henry Segerman. Triangulating the figure 8 knot complement.