Capturing Hyperbolic Space using Virtual Reality

Joseph Frias, advised by Dr. Sean Lawton

Mason Experimental Geometry Lab, George Mason University

Abstract

Using the half-space model for hyperbolic geometry in dimension 3, the VR team at MEGL is working to visualize geodesics and provide a way to render hyperbolic manifolds in a virtual reality setting. Currently, geodesics and basic polyhedra have been rendered in the Unity game engine. The current work is focusing on building lattices out of these basic polyhedra.

Introduction to Hyperbolic Geometry

The half-space model for 3D hyperbolic geometry is, as a set, the upper half-space of $\mathbb{R}^3$. This set is endowed with a metric function

$$\text{dist}(\vec{x} = (x_1, x_2, x_3), \vec{y}) = \text{arc}\cosh(1 + \frac{r_{xy}^2}{2x_3y_3})$$

where r_{xy} is the Euclidean distance between the two 3-vectors. Given this structure, curves of minimizing distance between points are circles on the affine plane defined by the vector difference of the points, and the vertical direction.

As it is a metric space, we can find the group of homeomorphisms on it that also preserve the distance function. This group, the isometry group of the space, is isomorphic to $\text{PGL}(2, \mathbb{C}) \approx \text{PSL}(2, \mathbb{C})$. In particular, we can represent this isometry group by the extended Möbius transformations. This gives us an easy interpretation of the group action on $\mathbb{H}^3$. Realizing the upper half-space of $\mathbb{R}^3$ as

$$\{z + jt | z \in \mathbb{C}, t \in \mathbb{R}_+\} \subset \mathbf{H},$$

where $\mathbf{H}$ is the ring of quaternions, a member $\phi = \{\pm \left(\begin{smallmatrix} a & b \\ c & d \end{smallmatrix}\right)\}$ of the isometry group acts on $\mathbb{H}^3$ by

$$\phi(\xi := z + jt) = \frac{a\xi + b}{c\xi + d}$$

.

Our full purpose is to provide a method for visualizing hyperbolic 3-manifolds, so we are working to provide a method for visualizing fundamental domains. This is easy enough, as we just have to connect the vertices of the polytope with geodesic edges. Constructing a lattice of such polytopes is less straightforward. To do this, we had to find the hyperbolic equivalent of reflecting across a face of a polytope in Euclidean geometry. To do this, we constructed a quaternionic version of sphere inversion that provided what we needed.

Programming Hyperbolic Geometry

To begin, we needed a basic tool of geometry : the straight line between two generic points A and B. Note that in Unity, vectors are left-handed, and y in the code will mean the vertical component. $-z$ will be the standard y in euclidean geometry.

```
Vector3[] hgeo(float s, Vector3 A, Vector3 B){

  Vector3[] geo = new Vector3[25];

  Vector3 Avg = (A+B)/2;

  Vector3 z = Vector3.up;

  if (Mathf.Abs(A.x-B.x) < .001f && Mathf.Abs(A.z-B.z) < .001f){// If the two points are vertical, then the geo is just the straight line between them

    for (int i = 0; i < 25; i++) {

      float t = i/25f;

      geo [i] = A+t*(B-A);// Check that this makes a path from A to B

    }

  }
```

In order to create the geodesic, we use an affine plane that contains the average of the two points. We also determine the case where the two points are on top of each other. The straight line in that case is just that - a straight line.

```
if (A.y != B.y){// this and the next block are based on a basic geometric construction, using the plane made by the difference vector (A-B) and the y axis

  float c = -A.y;// Using vectors based at the vector Avg

  Vector3 C = Avg + c * z;

  for (int i = 0; i < 25; i++) {// rotate and sweep out the geodesic

    float t = i / 25f;

    float angle = -Vector3.Angle(B-C,A-C) * t;

    //float angle = 360 *t;

    geo [i] = C + Quaternion.AngleAxis (angle, Vector3.Cross(B-A,z)) * (A-C);

    geo[i] =geo[i]+ s*(A+t*(B-A)-geo[i]);

  }

}
```

In the case where both points are the same distance from the xy-plane, we don't need to consider the z component of $\vec{A} - \vec{B}$. Note that we stay "based" at $\frac{\vec{A}+\vec{B}}{2}$ until we need to consider the origin and plot a point of the geodesic.

```
if (A.y != B.y && !(Mathf.Abs(A.x-B.x) < .001f && Mathf.Abs(A.z-B.z) < .001f) ) {// same as before, but transformed because A-B is not horizontal

  Vector3 vv = B - A;

  Vector3 l = vv - (vv.sqrMagnitude / vv.y) * z;

  float c = -Avg.y/(l.y);

  Vector3 C = Avg+c*l;

  for (int i = 0; i < 25; i++) {

    float t = i / 25f;

    float angle = -Vector3.Angle(B-C,A-C) * t;

    //float angle = 360 *t;

    geo [i] =C+ Quaternion.AngleAxis (angle, Vector3.Cross(vv,z)) * (A-C);

    geo[i] =geo[i]+ s*(A+t*(B-A)-geo[i]);

  }

}
```

Here is the process in general. We need to weight the z component of the difference, so that we actually find the center on the xy-plane.

More Unity

```
// mobius transformation
  Vector3 lFrac(Vector3 U, complex a, complex b, complex c, complex d){
        float x = U.x;
        float y = - U.z;
        float r = U.y;
        complex z = new complex (x,y);
        float e1 = x*x+y*y+r*r;
        float c1 = c.len (2);
        float sc = c1 * e1 + d.len (2) + 2 * (c * z*(~d)).x;
        sc = 1f / sc;
        complex w = new complex ();
        w = e1*a*(~c) + a*z*(~d)+ -(c*z*(~b))+b*(~d);
        w = sc * w;
        Vector3 V = new Vector3 (w.x,r*sc,-w.y);
        return V;
  }
```

We also wanted to code möbius transformations, so that we could work with the isometry group. To do this, we created a class of complex numbers, and worked out the complex and j component of the transformation.

```
// This is the map from the half plane to the ball
    Vector3 Ball(Vector3 V){
          Vector3 s = Vector3.zero;
          float ph1 = (1f-V.sqrMagnitude/4f)/(1f+V.sqrMagnitude/4f);
          float ch1 = ph1 + 1;
          float J = V.y;
          float t = J * ch1 / 2f;
          k = 1f/(ch1);
          s.x = V.x*k*ch1/2f;
          s.y = ph1*k;
          s.z = (V.z) * ch1*k / 2f;
          return s;
    }
```

We also included a map from the half-plane to the ball model of hyperbolic geometry.

```
Vector3 Sphere(Vector3 B, Vector3 A, Vector3 C){
      Vector3 S =  Vector3.zero;
      Vector3 a = new Vector3(A.x,-A.z, A.y);
      Vector3 b = new Vector3 (B.x, -B.z, B.y);
      Vector3 c = new Vector3 (C.x, -C.z, C.y);
      float bxy = (a.sqrMagnitude-b.sqrMagnitude)/2f;
      float byz = (c.sqrMagnitude-b.sqrMagnitude)/2f;
      matrix D = new matrix (2, 2);
      D.Mat [0, 0] = (a - b).x;
      D.Mat [0, 1] = (a - b).y;
      D.Mat [1, 0] = (c - b).x;
      D.Mat [1, 1] = (c - b).y;
      matrix DI = matrix.Inv (D);
      matrix l = new matrix (2, 1);
      l.Mat[0,0] = bxy;
      l.Mat [1, 0] = byz;
      matrix r = DI*l;
      float rx = r.Mat[1,0];
      float ry = r.Mat[0,0];
      S.x = rx;
      S.z = ry;
      S.y = (S - a).magnitude;
      return S;
}
```

In order to construct lattices of polytopes, we need some method of reflecting across planes. To do this, we found a way of finding a sphere, centered on the xy-plane, that intersects three specific points. The basic problem is that we get a set of two linear equations in the x and y components of the center (note these are the only nonzero components), because of the fact that all the points on the sphere (which include the three points we want on it) are equidistant from the center. We turn it into a matrix equation, and solve for the center.

App

We created a little application that showcases some of the geometry we've been working on. Go to https://github.com/jfrias3/PosterPres

Acknowledgements

We would like to thank Dr. Lawton for his help, and Dr. Christopher Manon and the NSF for funding us.