Modelling the relationship between a hyperbolic tessellation and a corresponding triply periodic polyhedron

A THESIS
SUBMITTED TO THE FACULTY OF THE GRADUATE SCHOOL
OF THE UNIVERSITY OF MINNESOTA
BY

Nirav Sharda

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
MASTER OF SCIENCE

Dr.Douglas Dunham

July 2016

Acknowledgements

I would like to take this opportunity and thank all the people without whose support and guidance, it would not have been possible to complete this thesis.

Firstly, I would like to thank my advisor, Dr. Douglas Dunham for his constant support and guidance throughout the course of my thesis.

I would like to thank my co-advisor Dr. Peter Willemsen for his support and guidance. I would also like to thank my defense committee member Dr. Marshall Hampton for his encouragement and support. I would take this opportunity to thank my graduate instructors Dr. Hudson Turner, Dr. Ted Pedersen and Dr. Henry Wang for sharing their knowledge and expertise.

I would like to thank my fellow thesis members Vamsidhar Kasireddy and Swetha Naidu, for their ideas, discussions and constant support.

Finally, I would like to thank my parents and my brother for their support, love, encouragement and motivation over the years, without which this would have not been possible.

Dedication

Dedicated to

my mother,

Dr. Sudha Sharda

my father,

Dr. Vinod Kumar Sharda,

and my brother,

Gaurav Sharda

Abstract

The process of creating artistic patterns has existed for a long time. Many great artists have contributed to this field. Amongst them are notably M.C. Escher, who made many major contributions to this field. He created a few hyperbolic tessellations, which are repeating patterns in hyperbolic geometry and are represented in the Euclidean plane, as formulated by some mathematicians. Creating these patterns was a very complex process, as it had to be done by hand. After his contributions, some programmers developed applications to create these repeating hyperbolic tessellations using the C, C++, and Java programming languages.

The motivation for my thesis is to create a simple and interactive tool that allows the user to create these hyperbolic tessellations. The user will have the ability to select the type of hyperbolic tessellation. The user will also be able to create triply periodic polyhedrons in this system. Finally, the system models the relationship between a hyperbolic tessellation and a corresponding triply periodic polyhedron using an interactive and simple mechanism. The tool was created in the Unity Game Engine using the C# programming language.

Contents

List of Figures vi

1 Introduction 1

2 Euclidean and Non-Euclidean Geometries 3

2.1 Euclidean Geometry . . . 3

2.2 Non-Euclidean Geometry . . . 4

2.2.1 Hyperbolic Geometry . . . 5

2.2.2 Spherical Geometry . . . 6

3 Models for Representing Hyperbolic Geometry 8

3.1 Beltrami-Klein Model . . . 8

3.2 Poincaré Model . . . 10

3.3 Coversion of a Point between the Klein and the Poincaré Model . . . 11

3.4 Weierstrass Model . . . 11

3.5 Isomorphism Between Hyperbolic Models . . . 13

3.5.1 Weierstrass-Klein Model . . . 13

3.5.2 Weierstrass-Poincaré Model . . . 13

4 Implementation 15

4.1 Tessellation 15
4.1.1 Regular Tessellation 15
4.1.2 Semi-regular Tessellation 16
4.1.3 Non-regular Tessellation 17
4.2 Hyperbolic Tessellation 17
4.3 Mesh Generation in Unity 18
4.4 Generation of Hyperbolic Tessellation in Unity 20
4.4.1 Finding vertices for all the hyperbolic polygons 20
4.4.2 Generation of Hyperbolic Lines 22
4.4.3 Assigning Texture or Random Color to Polygons 24
4.5 Generation of Triply Periodic Polyhedron in Unity 25
4.6 Relation Between Hyperbolic Tessellation and Triply Periodic Polyhedron . 29
4.6.1 Modelling the relation with Line Renderer 29
4.6.2 Modelling the relation with Mouse Position 30

5 Graphical User Interface **34**

6 Results **38**

7 Conclusions **44**

8 Bibliography **45**

List of Figures

1.1 The {4,6} hyperbolic tessellation and the {4,6|4} polyhedron generated by the program 2

2.1 Hyperbolic Axiom 5
2.2 Hyperbolic Triangle 6
2.3 Great cicle and antipodal points 7

3.1 Points in Klein Model 9
3.2 Lines in the Klein Model 9
3.3 Lines in Poincaré Model 10
3.4 Hyperboloid of one sheet 12
3.5 Projection from the Hyperboloid to Poincaré model 14

4.1 Regular Tessellations 16
4.2 Semi-regular Tessellations 17
4.3 A {6,4} hyperbolic tessellation with random color polygons 18
4.4 A square generated in unity 19
4.5 Fundamental Polygon and its circumscribing circle 21
4.6 Inverse of a point on the fundamental polygon 22
4.7 Before subdivsion 23

4.8 After subdivision 24
4.9 The {4,6|4} triply periodic polyhedron 26
4.10 A Square face of the {4,6|4} polyhedron 27
4.11 Angels and Demon pattern 27
4.12 A block of the {4,6|4} triply periodic polyhedron 28
4.13 Modelling the Relationship with Line Renderer 30
4.14 Modelling the Relationship with Sphere 31

5.1 Starting Screen of the application 34
5.2 Options to setup the hyperbolic tessellation type 35
5.3 Options to select the type of triply periodic polyhedron 36
5.4 Split-screen showing the hyperbolic tessellation and the corresponding triply periodic polyhedron 37

6.1 A {4,6} hyperbolic tessellation with 3 layers and randomly colored polygons 39
6.2 A {6,6} hyperbolic tessellation with 3 layers and randomly colored polygons 40
6.3 A {8,3} hyperbolic tessellation with 3 layers and randomly colored polygons 41
6.4 A {10,3} hyperbolic tessellation with 3 layers and randomly colored polygons 42
6.5 A {10,3} hyperbolic tessellation with 4 layers and randomly colored polygons 43

1 Introduction

Drawing artistic geometric patterns has been in existence for a long time. A lot of different civilizations across various regions have incorporated these artistic patterns, which is evident from the ruins of these civilizations. Some civilizations such as the Arabic, Chinese, Sumerians, Egyptian, Japanese, Persians, Greek and Romans have inscribed these artistically repeating patterns in the architecture of their floors, walls, buildings and also the ceilings, where these were made of clay. These designs had some symmetrical and asymmetrical aspects associated with them [7].

Many mathematicians in the past have done a lot of work trying to understand these geometric patterns. They eventually came up with algorithms to describe these geometric patterns. They have used Euclidean geometry and non-Euclidean geometries to represent these patterns.

Amongst these works are the notable contributions of M.C Escher, a very famous Dutch artist. His major contribution was to represent an infinite tessellation on a 2-dimensional plane. Some of his work was focused on representation of a hyperbolic tessellation on a 2-dimensional plane. These hyperbolic tessellations were eventually give the notation of {p,q}, also known as the Schläfli symbol named after a great mathematician Ludwig Schläfli. Here “p” denotes a p-sided regular polygon and “q” of them meet at each vertex [4].

This idea can be extended into 3-dimensional space to get an infinite skew polyhedron. An infinite skew polyhedron is a 3-dimensional structure with non-planar figure, consisting of regular polygon faces and stretches infinitely into all 3-directions. These are sometimes

called hyperbolic tessellations, due to the presence of negative angle defect at each of their vertices. They are represented by the modified Schläfli symbol {p,q|r}, which was defined by H.S.M. Coxeter. Here "p" denotes a regular p-gon face, where "q" of them meet at each vertex, and they have regular "r" sided polygonal holes between them [2]. Figure 1.1 shows a {4,6} hyperbolic tessellation and a {4,6|4} polyhedron with angels and demons pattern generated by the program.

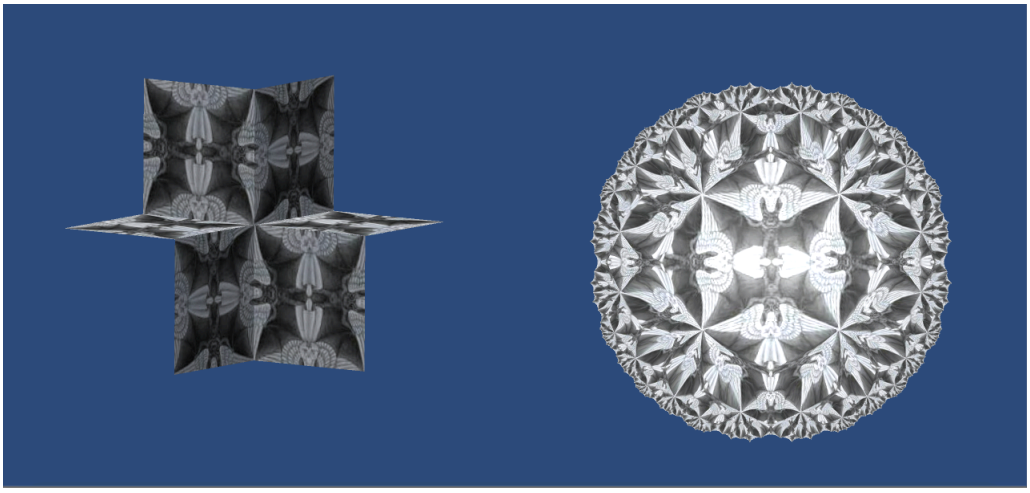

Figure 1.1: The {4,6} hyperbolic tessellation and the {4,6|4} polyhedron generated by the program

The are some geometrical symmetries between these hyperbolic tessellations and infinite skew polyhedron as observed by Dr.Dunham in his paper [4]. Previously some applications have been created to generate hyperbolic tessellations using C, C++ and Java. The idea for my thesis is to create a software that generates these hyperbolic tessellations and infinite skew polyhedrons using the C# programming and the Unity game engine. The system will have simple interacitve ability to create these hyperbolic tessellations and infinite skew polyhedrons. Finally the tool allows for a simple and interactive mechanism that models the relationship between hyperbolic tessellation and triply periodic polyhedra.

2 Euclidean and Non-Euclidean Geometries

"Geometry" is a branch of mathematics, and this word originated from the Greek words "gē" meaning earth and "metria" meaning to measure. This branch is associated with the study of shapes and sizes in all dimensions. A lot of work in this field was done by the great mathematician "Euclid" also known as the "father of geometry". His work is considered as the foundation of geometry. Geometry has progressed a lot and modern geometry consists of more complex areas like the study of differential and gravitational fields [8].

These geometries can be divided into different kinds. Amongst them Euclidean and non-Euclidean geometries have been used for the creation of repeating tessellations. The Euclidean geometry is also known as classical geometry is the most commonly known geometry. Amongst the non-Euclidean geometries, mainly spherical and hyperbolic geometries are used for these tessellations.

2.1 Euclidean Geometry

Euclidean geometry is the most commonly used and also the most well-known geometry. Most of it is found in "The Elements", a collection of books written by the great Greek mathematician Euclid around 300 B.C. The book mostly consisted of theoretical concepts, although nowadays his geometrical ideas can be used to solve hundreds of practical problems. The basis for his theories rely on five main axioms listed below:

1. Given two different points A and B, we can get a unique line m that passes through both these points.
2. Given any line-segments MN and RS, MN can be extended to NO, where NO is congruent to RS.
3. Given any line-segment, a circle with radius equal to the length of the line-segment can be constructed around it, and here one endpoint of the line-segment becomes the center of the circle.
4. Given any two right angles, they are always congruent.
5. The final postulate is also called the parallel postulate. It states that given any line m and a point S, not on that line, we can get only one unique line n which passes through S and is parallel to m.

2.2 Non-Euclidean Geometry

The foundation of non-Euclidean geometry was laid by Carl Friedrich Gauss, who investigated the parallel postulate problem. It was evident from his diary and personal letters to friend that he discovered a geometry that was completely different from Euclidean geometry. But, Gauss did not publish his findings in any paper. Later Janos Bolyai, independently came up with the discovery of non-Euclidean geometry, which was an appendix in his father's book. His father sent Gauss a letter regarding his son's discovery as he knew him well. Gauss in his reply back, informed Bolyai's of his own research in this direction. On receiving this reply Janos thought that Gauss had stolen his ideas. Janos didn't publish anything after this incident.

The first person to discover non-Euclidean geometry documental by publication, is Nikolai Lobachevsky, a great Russian mathematician. Lobachevsky's publication came to Gauss's attention and later he also showed his contributions to this field[1].

2.2.1 Hyperbolic Geometry

The major development of this field can be attributed to the great works by Gauss, Janos Bolyai and Lobachevsky. In this geometry all the axioms of Euclidean geometry hold, with the exception of the parallel postulate. This axiom is replaced by the hyperbolic axiom, which states that: If we are given a line m and a point S, which is not on that line m, we can get at least two distinct lines n and p, passing through the point S and both these lines are parallel to m.

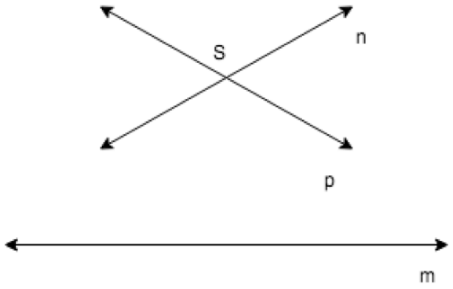

Figure 2.1: Hyperbolic Axiom

A few more important properties of the hyperbolic geometry are listed below:

1. Rectangles don't exist in hyperbolic geometry due to the hyperbolic axiom.
2. The sum of all the three angles of a triangle is less than 180 degrees.
3. The sum of all the angles of a convex quadrilateral is less than 360 degrees.
4. If there exist two similar triangles in hyperbolic geometry, then they are congruent [6].

These properties are not very intuitive to us, since we are used to a lot of Euclidean geometry from our childhood. The notion that the sum of all angles of a triangle is less than 180 degrees is something that doesn't look natural to us. The figure below shows a hyperbolic triangle with the sum of all angles being less than 180 degrees.

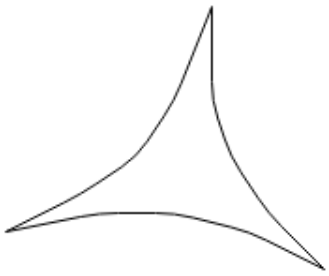

Figure 2.2: Hyperbolic Triangle

A few models to represent hyperbolic geometry has been discovered. Some of them include the Poincaré Disk, the Weierstrass and the Klein models. More details about these models can be found in the next chapter.

2.2.2 Spherical Geometry

Spherical geometry is also referred to as Riemannian geometry, after the name of its founder Bernhard Riemann. Spherical geometry is not a neutral geometry, because it doesn't have parallel lines. All the first four axioms of Euclidean geometry hold here, but the parallel postulate changes. If we are given any line m and a point P not on the line, there exists no lines parallel to the m. In a spherical geometry lines are represented by great circles. Take a plane passing through the center of the sphere and any two points on the sphere. The intersection of the plane with the sphere, gives us a great circle. Figure below shows a great circle E, which passes through the center O and two points N and S on the sphere. These points on the sphere N and S are also know as antipodal points [9].

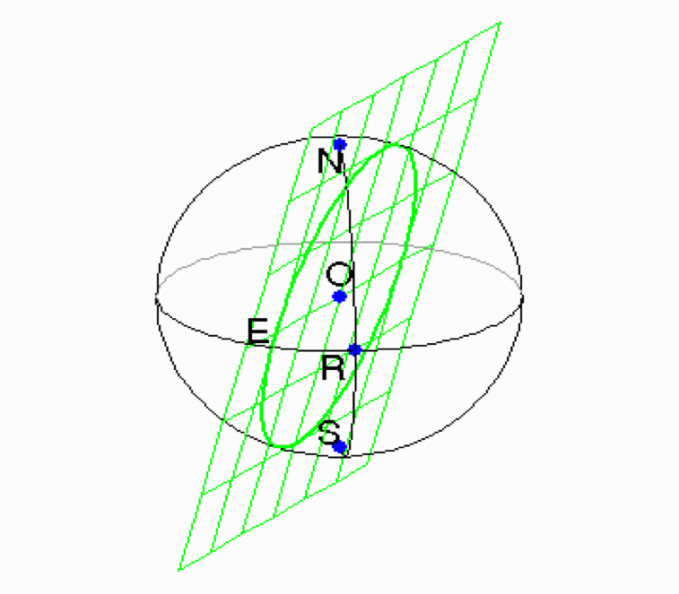

Figure 2.3: Great cicle and antipodal points

A few more important properties of the spherical geometry are listed below:

1. Since we don't have straight lines, three great circles intersect to form a spherical triangle. The sum of all the angles here is greater than 180 degrees and less than 540 degrees.

2. There are no parallel lines in spherical geometry, hence it is also called a non-neutral geometry.

3. Given two similar triangles, they are congruent as well.

3 Models for Representing Hyperbolic Geometry

The need for a model arises to represent all the objects and to prove all the axioms of the given hyperbolic geometry true. There are several models for representing hyperbolic geometry. These models can be broadly classified into two categories. One of them is a representation in the 2-dimensional Euclidean space and the other one is a representation in the 3-dimensional Euclidean space. The Beltrami-Klein model and the Poincaré Disk model are representations in 2-dimensional Euclidean space and are also finite hyperbolic geometry models. The Weierstrass model is a 3-dimensional Euclidean space representation of hyperbolic geometry and is also an infinite hyperbolic geometry model. All these three models and isomorphisms between them are discussed in detail in this chapter.

3.1 Beltrami-Klein Model

This model was proposed by a great German mathematician Felix Klein. This model is a finite representation of hyperbolic geometry on a 2-dimensional Euclidean circle. Given a circle with center O and radius OA, as seen in the Figure 3.1, all points B are considered to lie in the interior of the circle if OB<OA. The circle is also know as the bounding circle.

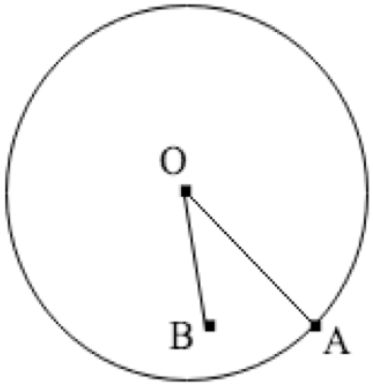

Figure 3.1: Points in Klein Model

In the Klein model all points in the interior of the circle like B, represent points on the hyperbolic plane. The definition of a line in this model is an open chord of the circle. An open chord here means all the points on a normal chord of the circle excluding the two end-points on the circle. Also we know that in hyperbolic geometry given a line and a point which is not on that line, we can get two lines passing through that point and parallel to the first given line. The figure below shows a line l and two lines (open-chords) parallel to l and passing through a point P.

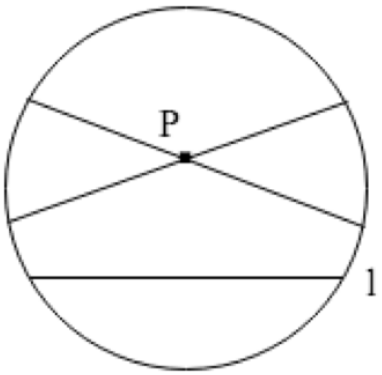

Figure 3.2: Lines in the Klein Model

3.2 Poincaré Model

This model was proposed by a great French mathematician Henry Poincaré. Similar to the Klein model, in this model all points on the hyperbolic plane are represented by points lying in the interior of a circle in the Euclidean 2-dimensional plane. This model is sometimes also referred to as the conformal disk model. Although we have the same points as we have in Klein model, the definition for lines in this model changes.

There are two type of lines in the Poincaré model. All the open chords passing through the center of the circle or open diameters are lines. Here open diameter refers to a diameter of a circle excluding the two end-points on the circle. The other lines are open arcs coming from another circle orthogonal to the bounding circle [6] [2]. Both these types of lines are shown in the figure below:

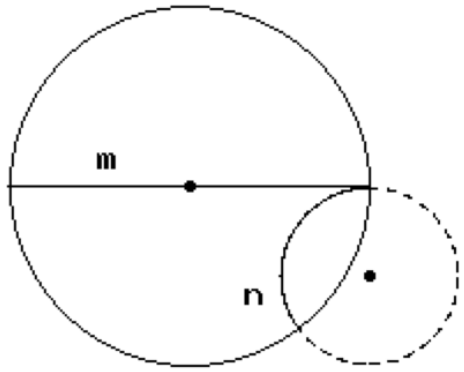

Figure 3.3: Lines in Poincaré Model

Compared to the Klein model, the greatest advantage of the Poincare model is that the congruence of angles is similar to that in Euclidean geometry [6].

3.3 Coversion of a Point between the Klein and the Poincaré Model

Both these models represent points on the interior of a bounding circle in Euclidean 2-dimensional space. This section gives the equations to convert a vector representing a point from one model to another.

Given a vector p representing a point on the Poincaré Disk model, the corresponding point k on the Klein model is given by the following equation below:

$$k = \frac{2p}{1 + p.p}$$

Also, given a vector k representing a point on the Klein model, a point p on the Poincaré model can be calculated by the following equation:

$$p = \frac{k}{1 + \sqrt{1 - k.k}} = \frac{(1 - \sqrt{1 - k.k})k}{k.k}$$

3.4 Weierstrass Model

This model is also commonly known as the hyperboloid model, since it is represented on a hyperboloid. This model is an infinite representation of Hyperbolic geometry. In this model all points on the hyperbolic plane are represented by points on the surface of the hyperboloid. The figure below shows a simple hyperboloid of one sheet.

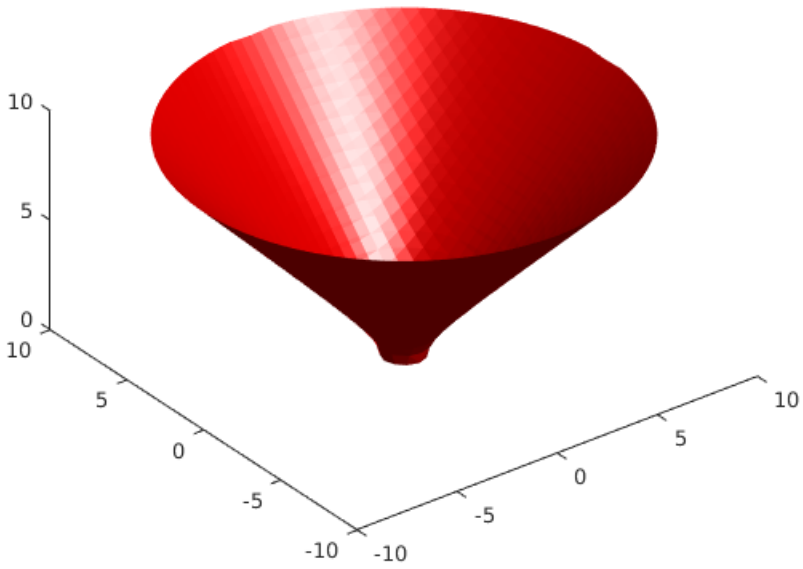

Figure 3.4: Hyperboloid of one sheet

Here the model only takes points on the upper part of the hyperboloid. This is a very important model and is very essential to derive the other models of hyperbolic geometry. The following equation represents a hyperboloid:

$$< X, X >= x^2 + y^2 - z^2 = -k^2$$

Here X is a vector in the 3-dimesnional Euclidean space (x, y, z). This gives two sheets of the hyperboloid: the upper and lower sheet. The model only uses the upper sheet as the lower one is a mere reflection of the upper one. The new mathematical representation of the upper sheet is as below:

$$< X, X >= -K^2 \; and \; (z > 0)$$

All these hyperbolic models have isomorphisms between them. These models are isomorphic since the points and lines in one model have corresponding similar points and lines in another model and they also preserve properties such as incidence and congruence [6]. The next section talks about these isomorphism relations in greater detail.

3.5 Isomorphism Between Hyperbolic Models

3.5.1 Weierstrass-Klein Model

The Klein model is obtained by projecting the hyperboloid through the origin onto the z=1 plane [2]. The equations for that are given below:

$$< x, y, x > \rightarrow < \frac{x}{z}, \frac{y}{z}, 1 >$$

The reverse from Klein to Weierstrass model projection is given by:

$$< x, y, 1 > \rightarrow \frac{1}{1 - x^2 - y^2} < x, y, 1 >$$

3.5.2 Weierstrass-Poincaré Model

Similar to the Klein model, the Poincaré model can also be obtained by taking a projection from the hyperboloid. Here the projection is through the point (0, 0, -1). The following equation represent the projection:

$$< x, y, z > \rightarrow \frac{1}{z + 1} < x, y, 0 >$$

The reverse from Poincaré to Klein model is given by:

$$< x, y, 0 > \rightarrow \frac{1}{1 - x^2 - y^2} < 2x, 2y, 1 + x^2 + y^2 >$$

The figure below shows the projection of a line from the hyperboloid to the Poincaré model. The brown line is the line in the hyperboloid model and the red one is on the Poincaré model and the projection is through (0, 0, -1) as it can be seen in the Figure 3.4 [10].

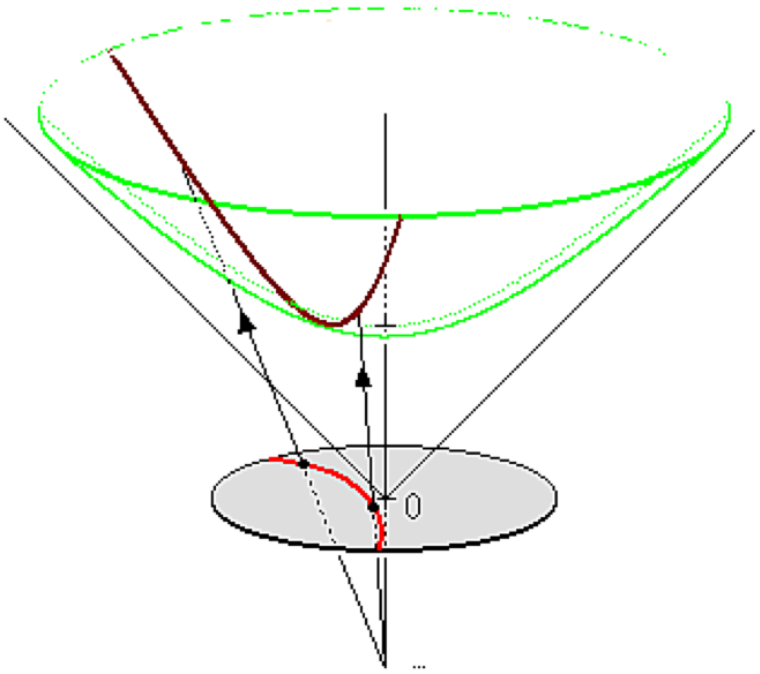

Figure 3.5: Projection from the Hyperboloid to Poincaré model

4 Implementation

This chapter will discuss about the generation of hyperbolic tessellation and triply periodic polyhedra in Unity-3D game engine. This chapter will also talk about how the relationship between a triply periodic polyhedron and the corresponding hyperbolic tessellation is modelled.

4.1 Tessellation

Tessellations are formed when a shape repeats itself over and over again on a plane with no gaps between them. The origin of the word "tessellate" can be traced back to the Greek word "tesseres", which means four. The early tilings were made up of four square tiles. There are three kinds of tessellations.

4.1.1 Regular Tessellation

A regular tessellation comprises of congruent copies of regular polygons repeating itself over a plane. All the angles of a regular polygon are equal and so are its sides. The figure below shows examples in the Euclidean plane.

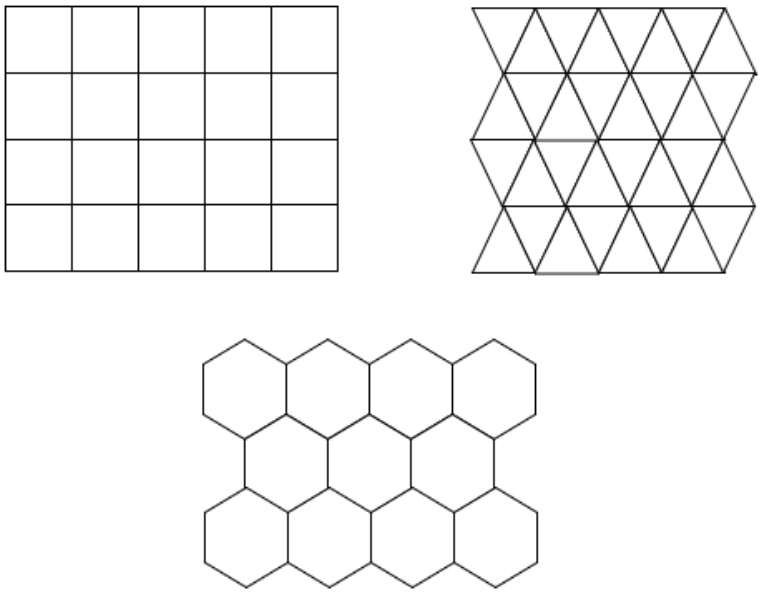

Figure 4.1: Regular Tessellations

4.1.2 Semi-regular Tessellation

A semi-regular tessellation is created when two or more kinds of regular polygons repeat itself over a plane. Any important property to notice here is that at each vertex, we get the same cyclic arrangement of regular polygons. The figure below shows some examples.

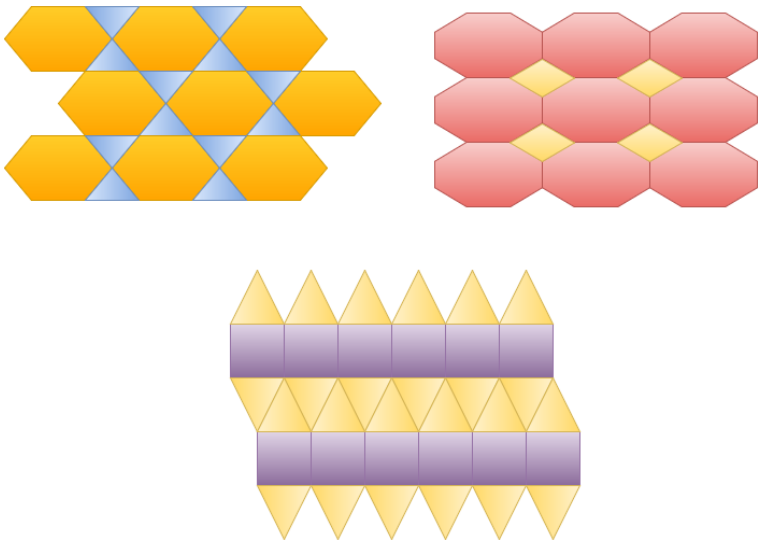

Figure 4.2: Semi-regular Tessellations

4.1.3 Non-regular Tessellation

A non-regular tessellation has regular polygons of different sides that repeat itself over a plane, and have no restriction on the order of polygons around a vertex.

4.2 Hyperbolic Tessellation

Once the idea of tessellation has been established, we can now talk about hyperbolic tessellations. Hyperbolic tessellations are repeating patterns in the hyperbolic plane. These patterns are also modelled in the finite Euclidean plane using the Poincaré and the Klein models. These are denoted by {p,q} notation. Here p denotes the sides of regular polygons, where q of them meet at each vertex. The condition for {p,q} to be a tessellation in the hyperbolic plane is $(p-2)(q-2) > 4$. The figure below shows a {6,4} tessellation generated by the application with polygons colored with random colors.

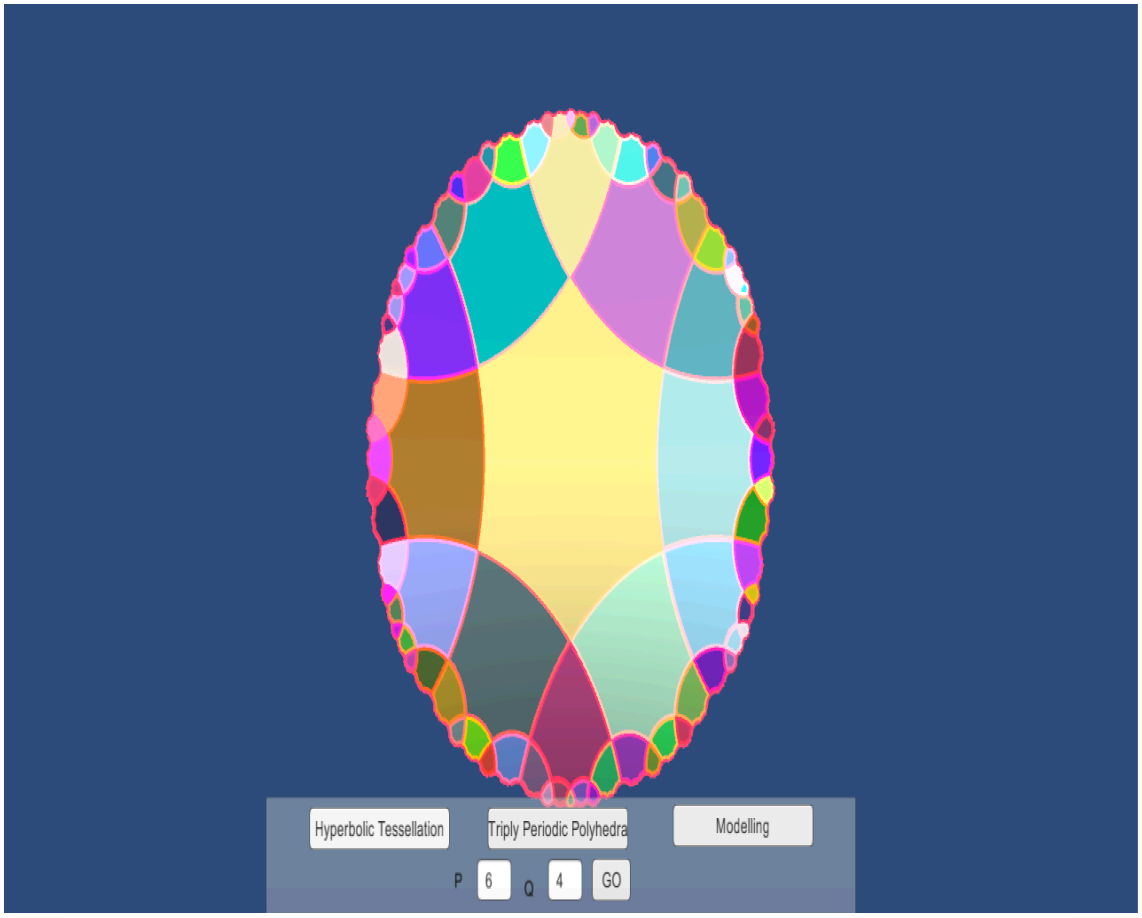

Figure 4.3: A {6,4} hyperbolic tessellation with random color polygons

4.3 Mesh Generation in Unity

Everything is represented by a gameobject in Unity. A gameobject refers to the most basic fundamental object in Unity that can represent a character, object, or an item. Every gameobject has a mesh associated with it. A mesh represents the vertices of the gameobject. Unity supports only two kinds of polygonal meshes: triangular and quadrilateral. To create a mesh through code in Unity, we have to assign vertices for the gameobject. Then we have to specify which vertices make a triangle, if we are using the triangular mesh. Finally some texture can be applied to this object using uvs, which defines the texture coordinates attached at each vertex. Also the normals can be defined which specifies the direction in

which the texture is rendered. This is explained in detail with an example. Suppose the user wants to create a square with a red colored texture in unity using an automatic mesh generation. The figure below shows a red square generated by a script. As it can be seen in the figure the blue lines around the square represent the outline of the mesh.

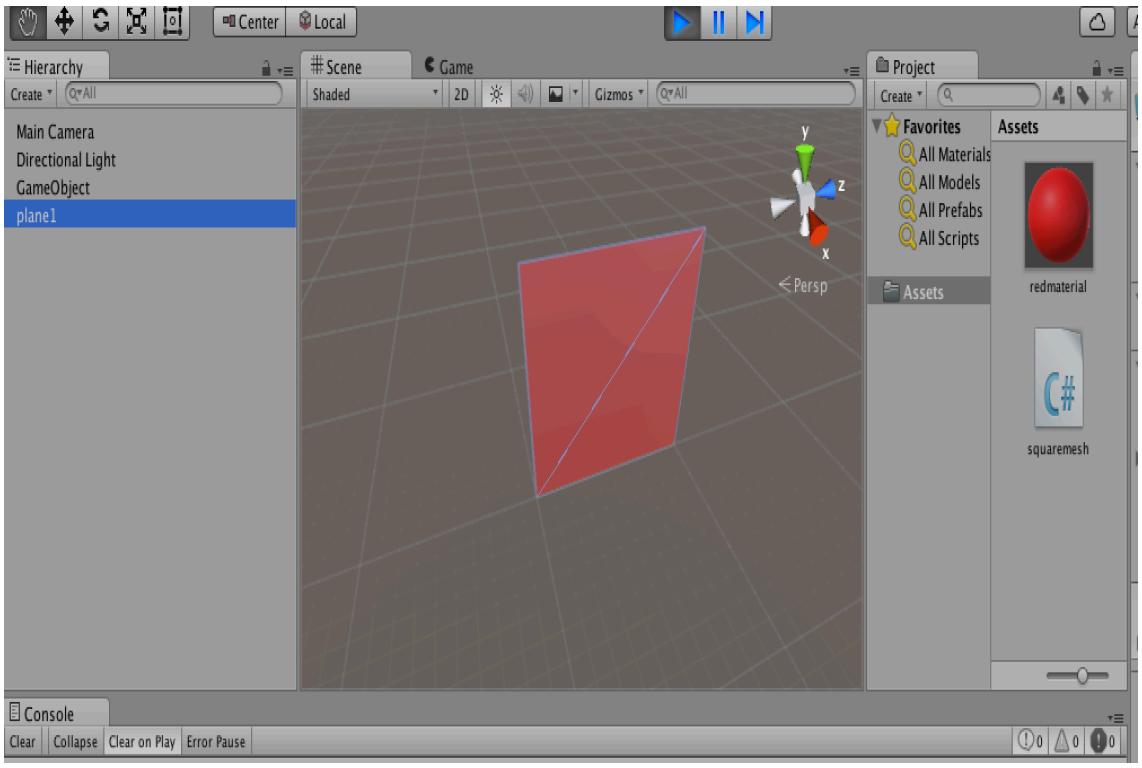

Figure 4.4: A square generated in unity

The first step is to assign vertices for the mesh. The vertices are stored in an array of Vector3. Vector3 is a data structure used in Unity to pass a 3-dimensional position in space. Here the array consists of 4 Vector3's namely (0,0,0) , (0,1,0) , (0,1,1), and (0,0,1). The next step is to assign triangles for the mesh, which is stored in an integer array. Here we have two triangles for the square. The first triangle has vertices 0,1,2 and the second triangle has vertices 2,3,0. An important thing to notice here is that these vertices are assigned clockwise. The final step is to add texture coordinates to the mesh which is defined by the uvs. The uvs are stored in an array of Vector2. Vector2 is a data structure used in Unity to

sometimes assign a 2-dimensional position or assign texture coordinates. These uvs here are simply (0,0), (1,0), (1,1), and (0,1). A uv has to be assigned for each vertex which assigns the texture coordinate to the actual 3-dimensional point.

4.4 Generation of Hyperbolic Tessellation in Unity

This subsection talks in details about the steps required to create a hyperbolic tessellation in Unity. The overview of the steps involved in the process include:

1. Creating an array of vertices for each hyperbolic polygon involved in the tessellation.
2. Generating hyperbolic lines between vertices for all the polygon obtained from the previous step.
3. Applying random colors or texture to all the polygons by setting up the uvs.

These three steps are discussed in detail in the subsections below.

4.4.1 Finding vertices for all the hyperbolic polygons

The method that is used to obtain the vertices for all the polygons is heavily reliant on the concept of inverse geometry. The first step to get all the vertices for all the polygons, involves finding the vertices of the fundamental polygon i.e. the center polygon in the tessellation. This can be done with the help of a circumscribing circle as shown in the figure below.

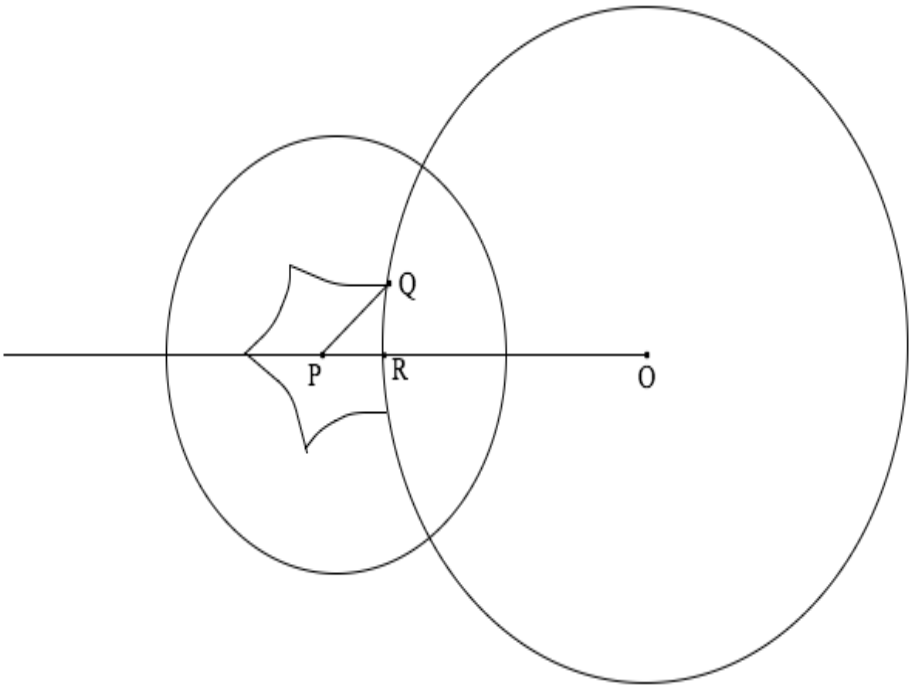

Figure 4.5: Fundamental Polygon and its circumscribing circle

The angle P in the figure is π/p,the angle Q is the figure is π/q and the angle R is $\pi/2$. Based on some simple trigonometric equation developed by Coxeter [3], the coordinates for point Q can be easily obtained. Now it is also known that all vertices have an angle of $2\pi/p$ between them. So the point Q can be rotated by an angle of $2\pi/p$ to get the coordinates of all the vertices of the fundametal polygon.

Now once we have all the vertices of the fundamental polygon, the next step involves obtaining vertices for all the polygons other than the fundamental polygon. This can be achieved by reflection of the obtained vertices across the hyperbolic lines. An important thing to observe here is that these hyperbolic lines are circular, so the reflections around these lines is just the inverse of a vertex with respect to the circumscribing circle. The figure below explains this in a greater detail.

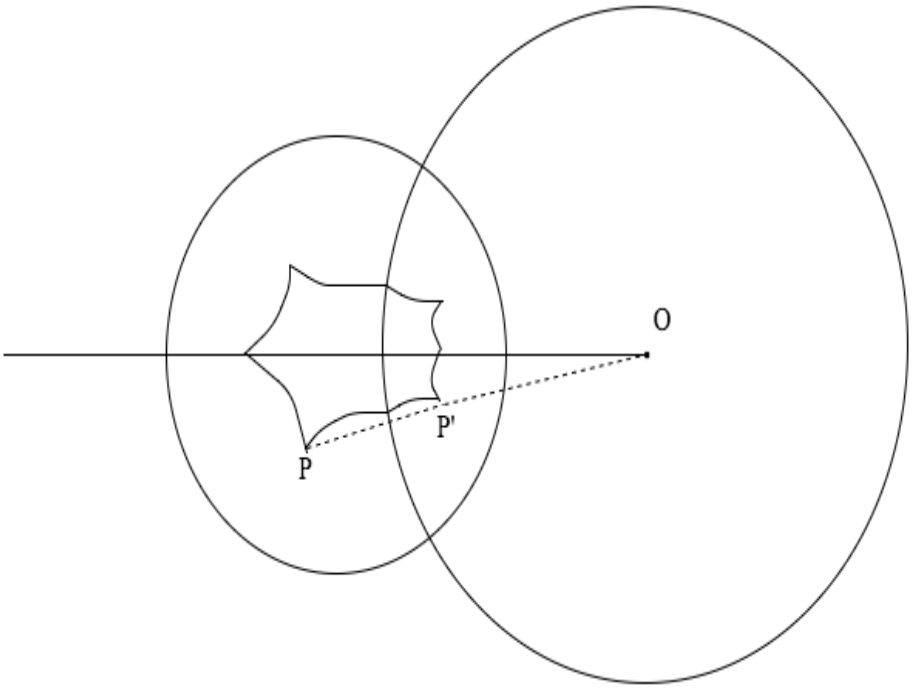

Figure 4.6: Inverse of a point on the fundamental polygon

The figure show the reflection of point P, a vertex of the fundamental polygon to get the vertex P' of a polygon in the next layer. Similarly if we do this reflection for other points of the fundamental polygon with respect to the hyperbolic line shown in the figure we get vertices for one polygon in the layer next to the fundamental polygon. If we keep doing this for other hyperbolic edges of the fundamental polygon, we can get vertices for other polygons in the next layer.

4.4.2 Generation of Hyperbolic Lines

Once we have all the vertices for all the polygons, the next step in the process is to generate hyperbolic lines between these vertices that we just obtained from the previous step. It is know that lines in the Klein model are straight lines. Also Unity renders meshes in the form of triangles. Now suppose we have to create a hyperbolic line between two Poincaré points obtained from the previous step. We can first convert these points to

Klein points. Now there is a straight line-segment between them. We can subdivide this line-segment in more points between them. Now when we convert all these points back to Poincaré points we have a hyperbolic line between the two original points. The below figures explain this in more detail.

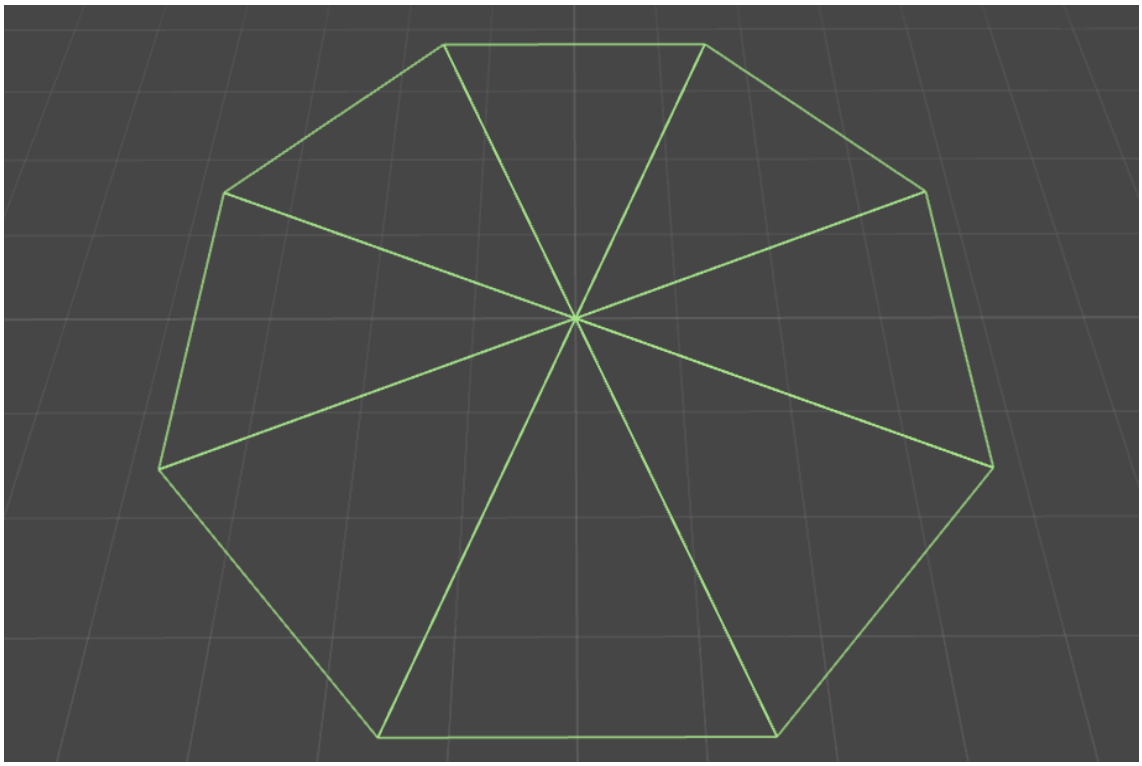

Figure 4.7: Before subdivsion

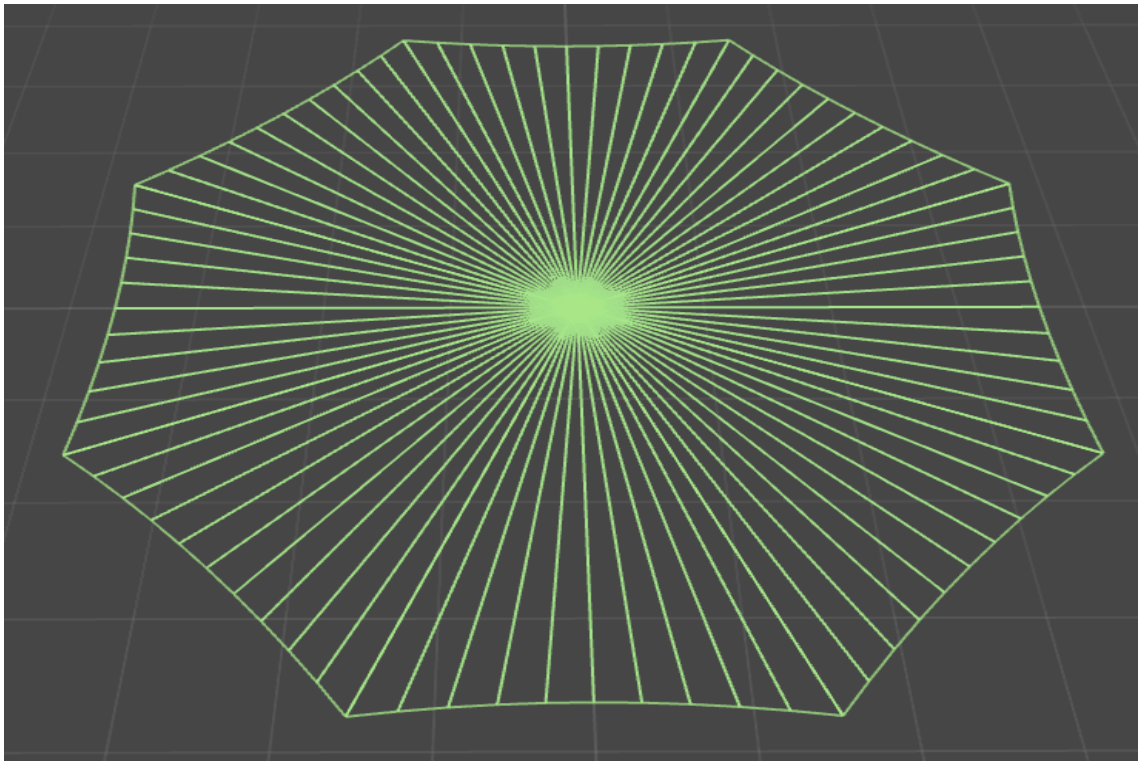

Figure 4.8: After subdivision

The first figure shows the stage where the vertices are converted from the Poincaré to the Klein model. In the Klein model we have a straight line-segment between two points as shown in the figure titled Before subdivision. Now it is very easy to divide a line-segment into more than 2 points between them. We divide each of the line-segments between every two pair of vertices of the polygon into 15 points between them. Now when these are converted back into Klein model we get hyperbolic lines betweeen them. This is evident in the Figure 4.8.

4.4.3 Assigning Texture or Random Color to Polygons

The final step is to assign a texture or add random color to each polygons. To assign a texture we have to assign texture coordinates to all vertices generated in step two after the subdivide step. Similar to the subdivision of vertices in the second step, a similar function

is used to set up these texture coordinates for all the vertices. Once these uvs are set up, the Unity game engine can easily attach texture to all the polygons desired. To assign random color to each of the polygons we can simply give random color to the gameobject's mesh renderer.

4.5 Generation of Triply Periodic Polyhedron in Unity

In geometry a polyhedron is defined as a solid figure consisting of many planar faces. A triply periodic polyhedron sometimes also referred to as infinite skew polyhedron is a 3-dimensional structure containing regular polygonal faces and stretches infinitely in all 3-dimensions. They are represented by the modified Schläfli symbol {p,q|r}. Here "p" denotes a regular p-gon face, where "q" of them meet at each vertex, and they have regular "r" sided polygonal holes between them. The basic idea in the creation of a triply periodic polyhedron in Unity is to create its regular polygon first, using the mesh generation technique already mentioned in a previous section. Then the idea is to replicate this basic building block at different positions and different angles to create the triply periodic polyhedron. This is explained in more detail using the {4,6|4} triply periodic polyhedron as an example. The figure below shows the {4,6|4} triply periodic polyhedron with the angels and demons pattern on it generated by the application.

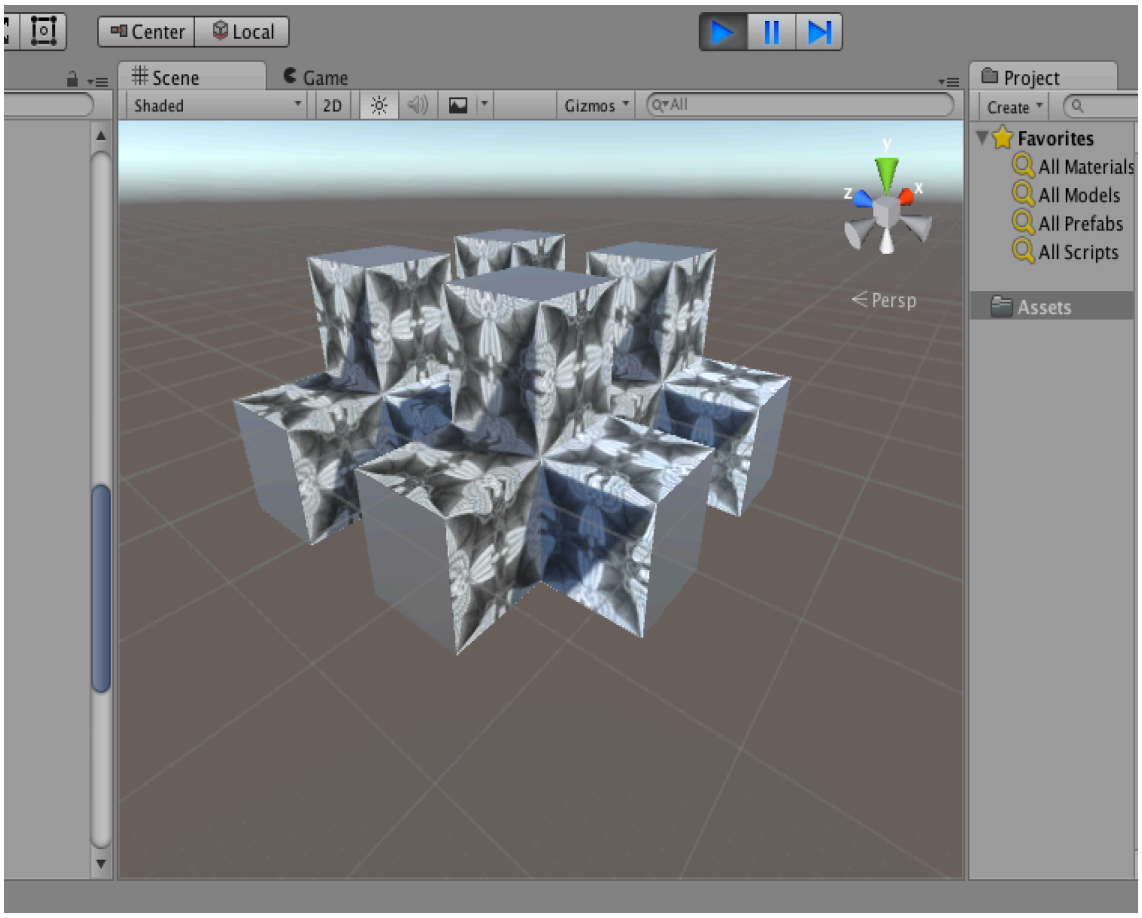

Figure 4.9: The {4,6|4} triply periodic polyhedron

The first step to create a {4,6|4} triply periodic polyhedron is to create its very basic face which is a square. Using the values of p,q i.e. 4 and 6, the Poincaré points of the centre tile in the hyperbolic tessellation are obtained from steps previously mentioned. These points when taken in the Klein model give the equivalent points for a face in the triply periodic polyhedron. Once we have the 4 vertices of the square face, we can easily obtain the centre of the square using a simple mathematical equation which is the sum of all the vertices divided by the number of vertices. Now the idea is to break this square into 4 triangles each containing two vertices and the centre of the square. Now we add all these points in the Vector3 array. The figure below explains this in much detail.

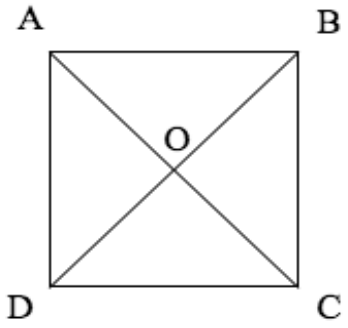

Figure 4.10: A Square face of the {4,6|4} polyhedron

The figure above shows a square with vertices A,B,C and D and the centre O. Once we have all these points stored in Vector3, we can create an array of Vector3 consisting of them. We can add them in the order of the triangles needed. The Vector3 array consists of A,B,O for the first triangle, then B,C,O for the second, C,D,O for the third and D,A,O for the fourth and last triangle. Once we have this vertex array set up for the mesh, we can assign the triangles for the mesh. Since we have to add the vertices in the order for the triangles to be setup easily, the triangles integer array in the mesh is just numbers 0 through 11. Now the final step is to assign a texture to it. The figure below show the angels and demons pattern which is attached to the triply periodic polyhedron.

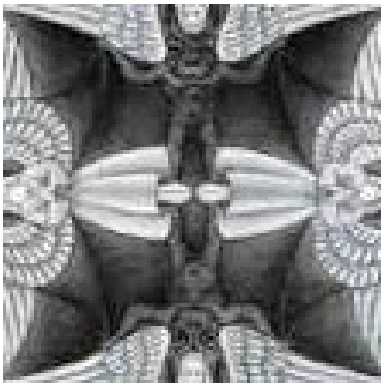

Figure 4.11: Angels and Demon pattern

The idea here while assigning the uvs to the mesh is to assign a texture coordinate for each vertex defined in the vertices array. So for points A,B and O in the square above these coordinates will be (0,1),(1,1) and (0.5,0.5) respectively. Also these texture coordinates i.e. uvs are stored in Vector2 array. Similarly it can be done for all other vertices. Once this is done we will have a square face for the {4,6|4} triply periodic polyhedron rendered in Unity. Now this face can be replicated, rotated and translated to generate a very basic block of the {4,6|4} triply periodic polyhedron. To replicate a gameobject in Unity the following function can be used.

```
1  Instantiate(gameobject_name, new_position, rotation);
```

Here the instantiate function takes in three arguments. The first argument is the gameobject name that is to be replicated, the next argument is a Vector3 which specifies the new position of the gameobject and the final argument in a Quaterion which is used in Unity to specify a rotation. Using the third argument we can specify the rotation associated with the new gameobject. The figure below shows such a block generated in Unity by code with one of the meshes highlighted in blue.

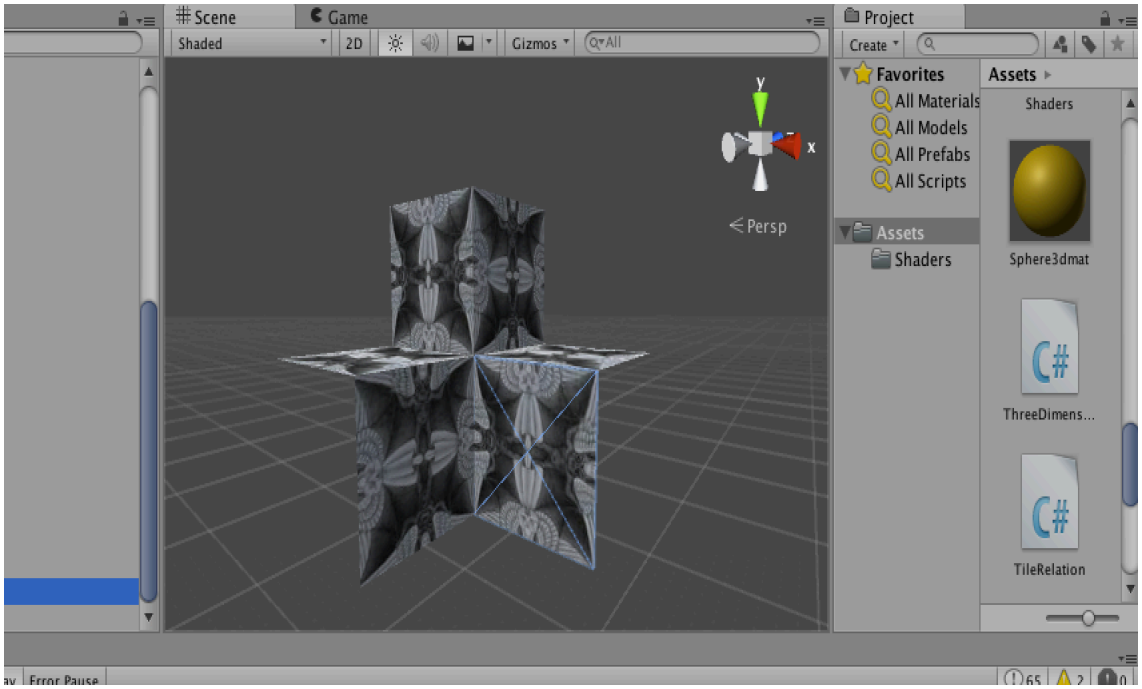

Figure 4.12: A block of the {4,6|4} triply periodic polyhedron

Also this block can be replicated and translated to create an infinite triply periodic polyhedron using the basic function specified before.

4.6 Relation Between Hyperbolic Tessellation and Triply Periodic Polyhedron

Polygons in hyperbolic geometry have a negative angle defect. This arises when the sum of all angles of a hyperbolic triangles is less than the usual 180 degrees in the Euclidean geometry, or when the sum of all angles of a hyperbolic quadrilateral is less than the usual 360 degrees. Dr. Dunham noticed the presence of these negative angle defects in the triply periodic polyhedrons, hence they can also be known as 3-dimensional hyperbolic tessellations [4]. This section aims to explain how this relationship has been modelled.

In the program we have two kinds of interactive mechanisms which clearly and very easily depict the relation between a hyperbolic tessellation and the corresponding triply periodic polyhedron. We represent them via the {4,6} hyperbolic tessellation and the {4,6|4} triply periodic polyhedron.

4.6.1 Modelling the relation with Line Renderer

The first mechanism which allows us to model the relationship between a hyperbolic tessellation and the corresponding triply periodic polyhedron, is using the line renderer available in Unity. A line renderer in Unity allows the user to add a line of variable width between any two user defined points. The figure below shows an image of the {4,6} hyperbolic tessellation and the {4,6|4} triply periodic polyhedron along with the line renderer.

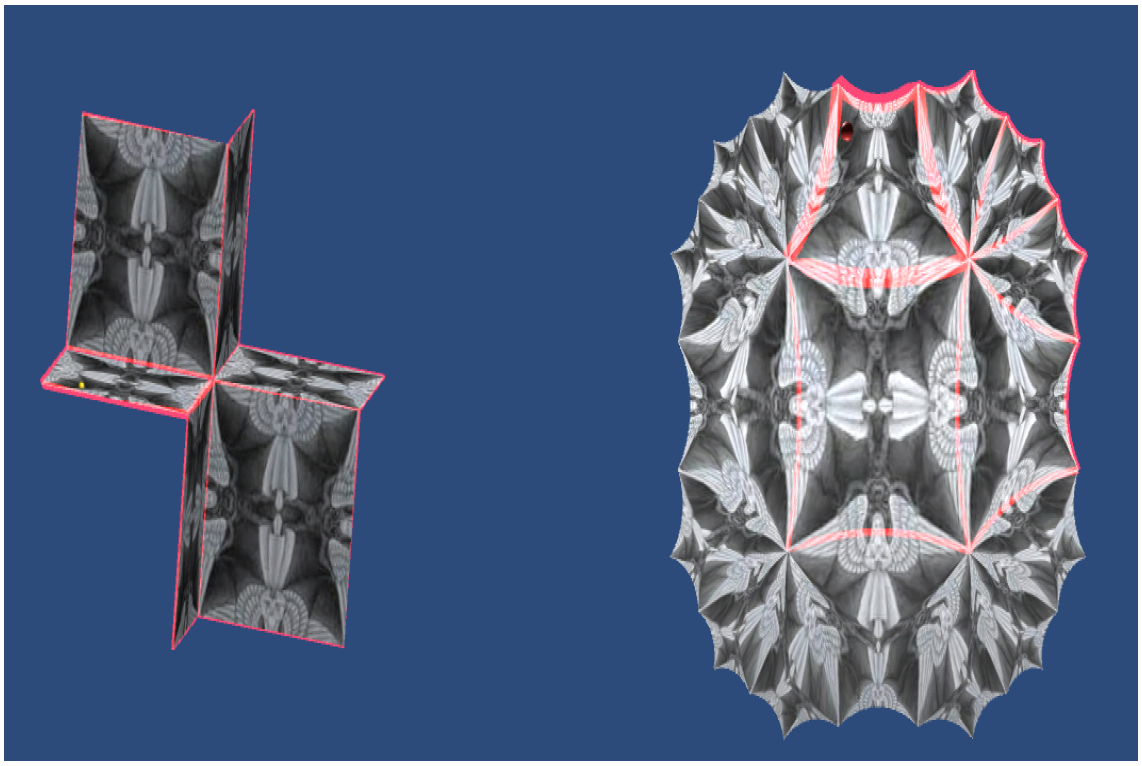

Figure 4.13: Modelling the Relationship with Line Renderer

As it is visible in the figure, the pink lines are generated via the linerenderer. Also the current polygonal face in the triply periodic polyhedron is highlighted based on the mouse position shown by the red sphere. Along with this polygonal face, the corresponding tile in the hyperbolic tessellation is also highlighted. The user has the ability to move the mouse pointer on the triply periodic polyhedron and based on the polygonal face selected by the mouse pointer location, the width of the linerenderer increases to highlight the selected polygonal face.

4.6.2 Modelling the relation with Mouse Position

The other mechanism which allows us to model the relation between a hyperbolic tessellation and the corresponding triply periodic polyhedron is with the help of a small sphere, whose position can be changed in the hyperbolic tessellation with the help of the mouse pointer. The figure below shows the red sphere in the hyperbolic tessellation and the corresponding yellow sphere in the triply

periodic polyhedron.

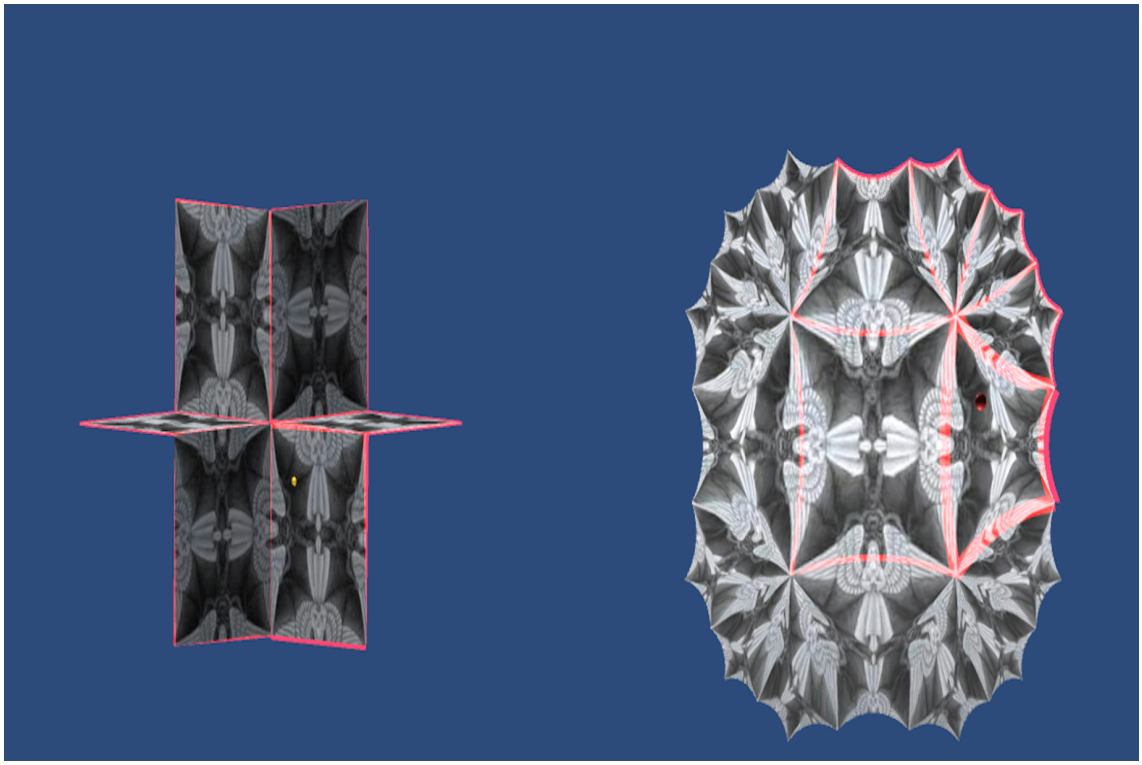

Figure 4.14: Modelling the Relationship with Sphere

The user has the ability to interactively move the red sphere in the hyperbolic tessellation. The user has to take the mouse pointer over the sphere to pick it. Now based on the position of the mouse pointer the sphere moves and this motion is only limited to the faces highlighted by the pink line renderer. The corresponding movement based on this is shown by the yellow sphere in the triply periodic polyhedron. To achieve this, some basic transformations have been applied which also depends on the tile in the hyperbolic tessellation.

For the center polygon we get the Poincaré point based on the position of the red sphere. Now if we convert this point to the corresponding Klein model point, we get the position of the point where the yellow sphere should be place in the triply periodic polyhedron. Finally, we have to select the appropriate tile and place the yellow sphere at the point given by the point in the Klein model.

Finding the corresponding point in the triply periodic polyhedron, for all the other polygons in the hyperbolic tessellation is slightly more complicated. The following steps give an overview to get

the equivalent representation in the triply periodic polyhedron for the hyperolic tessellation point.

1. First the point or the position of the red sphere which is in the Poincaré model is converted into a point in the Weierstrass model.
2. Now based on the polygon in which the point lies, a transformation is applied in this model to get the corresponding equivalent point in the center polygon in the Weierstrass model.
3. Now this point in transformed back into the Klein model, which gives us the position of the point from the center of the corresponding polygon in the triply periodic polyhedron, representing this hyperbolic polygon. This is the position where the yellow sphere is placed.

The transformations used in these steps include rotations around the edge of the fundamental polygon and around the hypotenuse of the fundamental polygon. These also include some basic rotations around the x,y and z axis. The following are these matrices.

$$Reflect\_Edge = \begin{bmatrix} -cosh2q & 0 & sinh2q \\ 0 & 1 & 0 \\ -sinh2q & 0 & cosh2q \end{bmatrix}$$

$$Reflect\_Hyptoenuse = \begin{bmatrix} cos(2\pi/p) & sin(2\pi/p) & 0 \\ sin(2\pi/p) & -cos(2\pi/p) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$Reflect\_EdgeBisector = \begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

The Reflect_Edge matrix when multiplied to a point, gives the reflection of that point with

respect to an edge of the fundamental polygon. The Reflect_Hypotenuse matrix when multiplied to a point, gives the reflection of that point with respect to the hypotenuse of the fundamental polygon. The Reflect_EdgeBisector matrix when multiplied to a point, gives the reflection of that point with respect to the edge bisector of the fundamental polygon. In these matrices the values of p and q come from the {p,q} hyperbolic tessellation. These matrices are obtained from the paper Creating Repeating Hyperbolic Patterns [5]. Some other rotations by θ are also performed. The matrix for that is specified below.

$$Rotate\_\theta = \begin{bmatrix} cos(\theta) & sin(\theta) \\ -sin(\theta) & cos(\theta) \end{bmatrix}$$

This matrix when multiplied to a Poincaré point give the final point rotated by θ degrees.

5 Graphical User Interface

The C# application created allows us to create different hyperbolic tessellations by specifying the values of p and q. It also allows us to create two kinds of triply periodic polyhedrons, the {4,6|4} polyhedron and the {6,6|3} polyhedron. The application also models the relationship between the hyperbolic tessellation and the triply periodic polyhedron via simple interactive features. This chapter gives some details about the Graphical User Interface associated with the application.

On starting the application, the user gets a screen with three buttons on it. One of the button is for the hyperbolic tessellation, the other is for the triply periodic polyhedron and the final button is the modeling button. The figure below shows the starting screen of the application.

Figure 5.1: Starting Screen of the application

When the user clicks the hyperbolic tessellation button, option to enter the values for p and q come up. The user has to enter desired values of p and q and click the Go button. When the go button is clicked the hyperbolic tessellation is generated with polygons filled with random colors. The figure below shows the screen when the hyperbolic tessellation button is clicked, with the options to add values of p and q for the {p,q} hyperbolic tessellation.

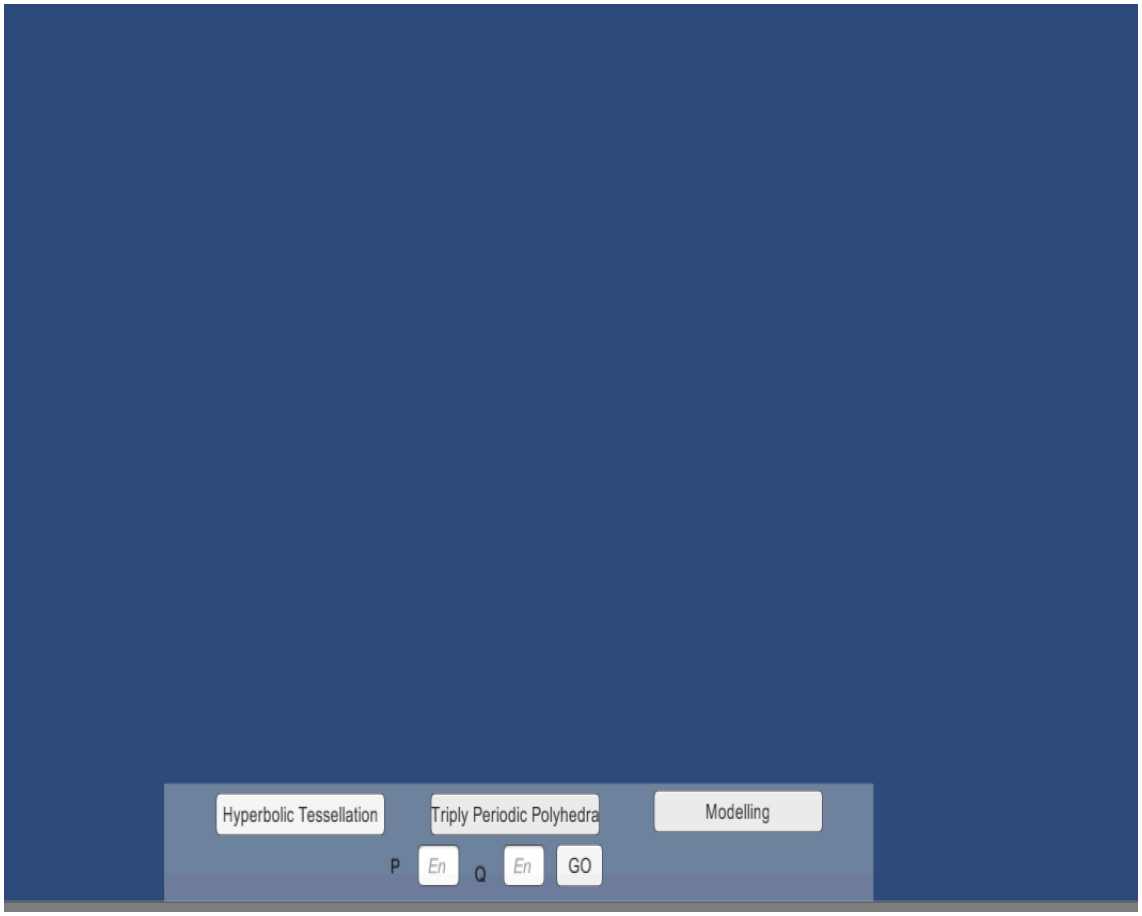

Figure 5.2: Options to setup the hyperbolic tessellation type

When the user clicks the triply periodic polyhedron button, two buttons to select the type of triply periodic polyhedron come up. The user can click on the type of triply periodic polyhedron desired, and the triply periodic polyhedron selected comes up with its polygonal faces colored with random colors. The figure below shows the screen when the triply periodic polyhedron button is clicked, with the options to select the type of triply periodic polyhedron.

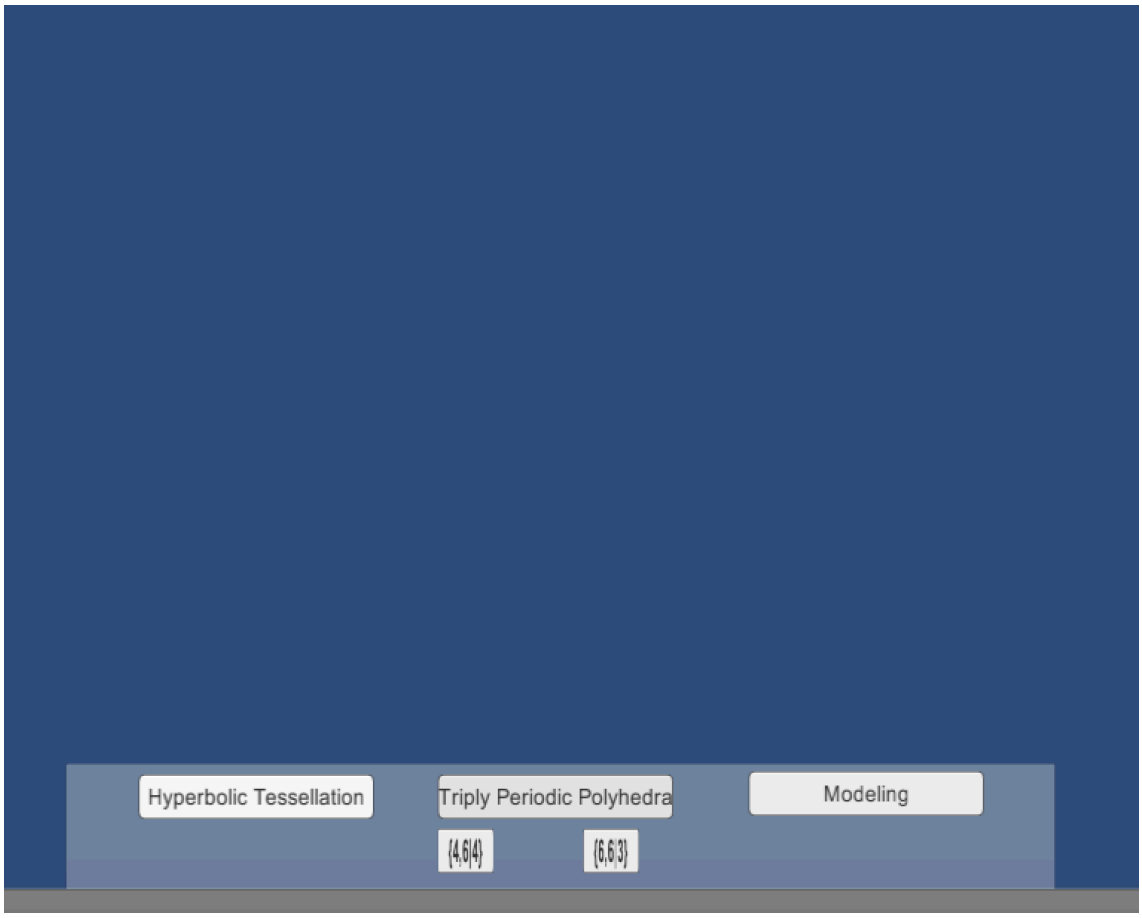

Figure 5.3: Options to select the type of triply periodic polyhedron

When the user clicks the modelling button, the application screen is split into two, one showing the {4,6} hyperbolic tessellation and the other showing the {4,6|4} triply periodic polyhedron, both with the angels and demons pattern on it. There are two simple mechanisms in the application that model the relationship between them, which are already mentioned in detail in the previous chapter. The figure below shows the split screen when the modeling button is clicked.

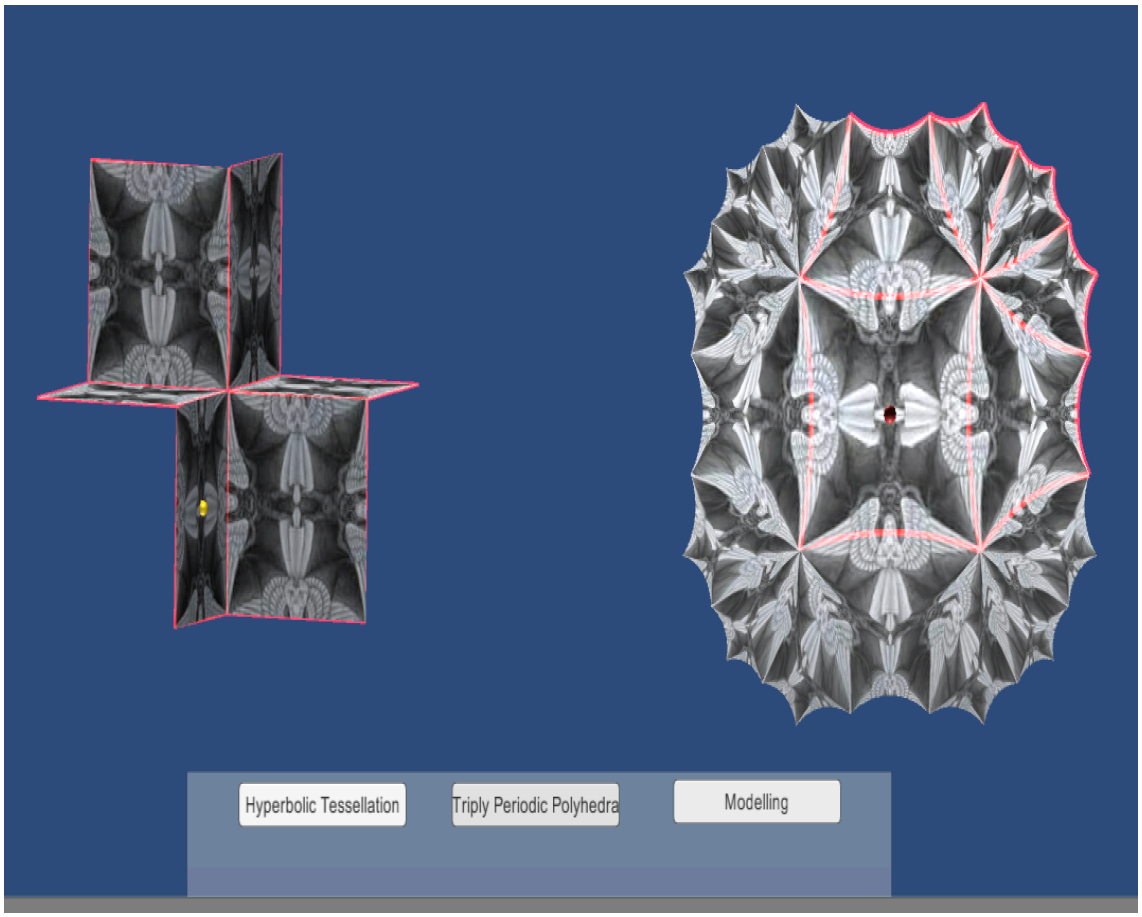

Figure 5.4: Split-screen showing the hyperbolic tessellation and the corresponding triply periodic polyhedron

6 Results

This section illustrates the results generated by the C# application created in Unity game engine. This application provides a simple interface to specify the values of p and q to generate a {p,q} hyperbolic tessellation. This application also allows the user to select some pre-defined triply periodic polyhedrons. Some applications have been created to generate hyperbolic tessellations using the C, C++, and Java programming language. This is the first application that allows the creation of both hyperbolic tessellations and triply periodic polyhedrons. The application also models the relation between a hyperbolic tessellation and a corresponding triply periodic polyhedron. The figures below show some screenshots of the hyperbolic tessellations and triply periodic polyhedrons generated by the application.

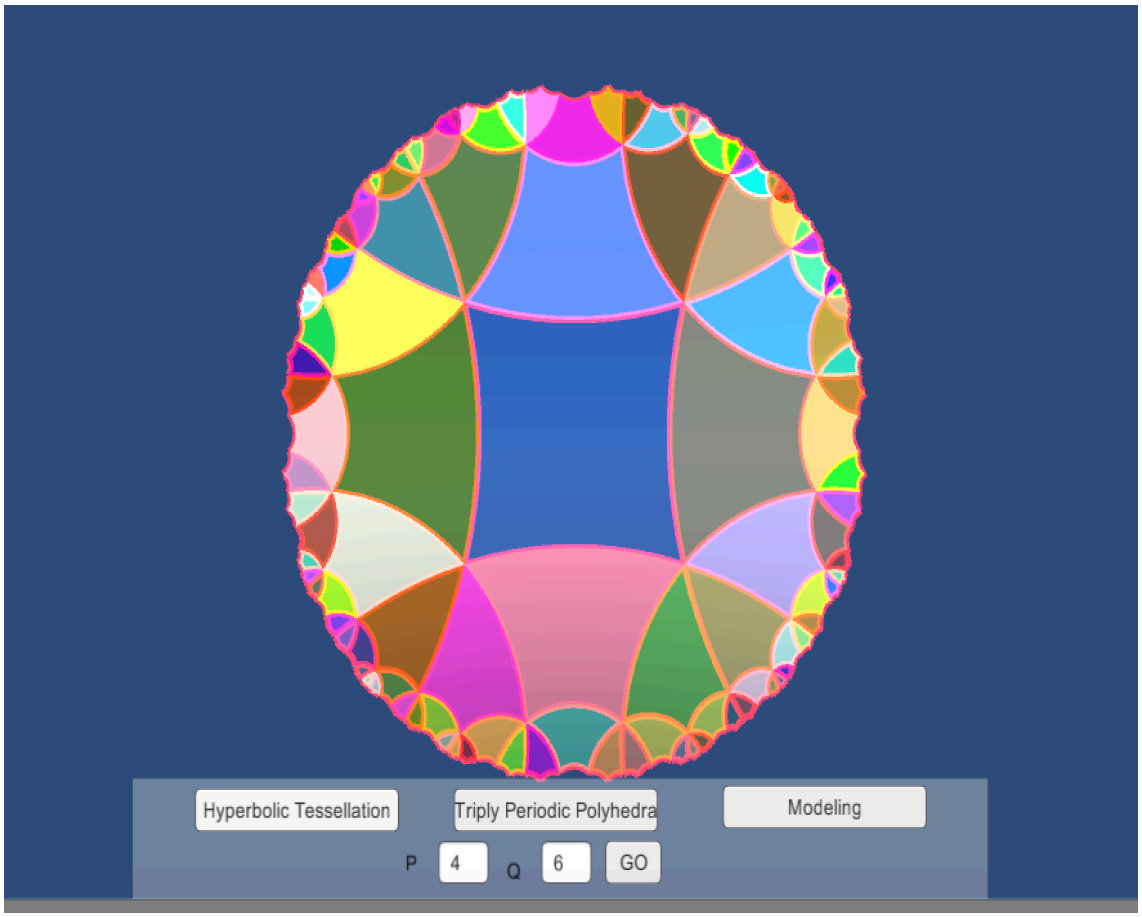

Figure 6.1: A {4,6} hyperbolic tessellation with 3 layers and randomly colored polygons

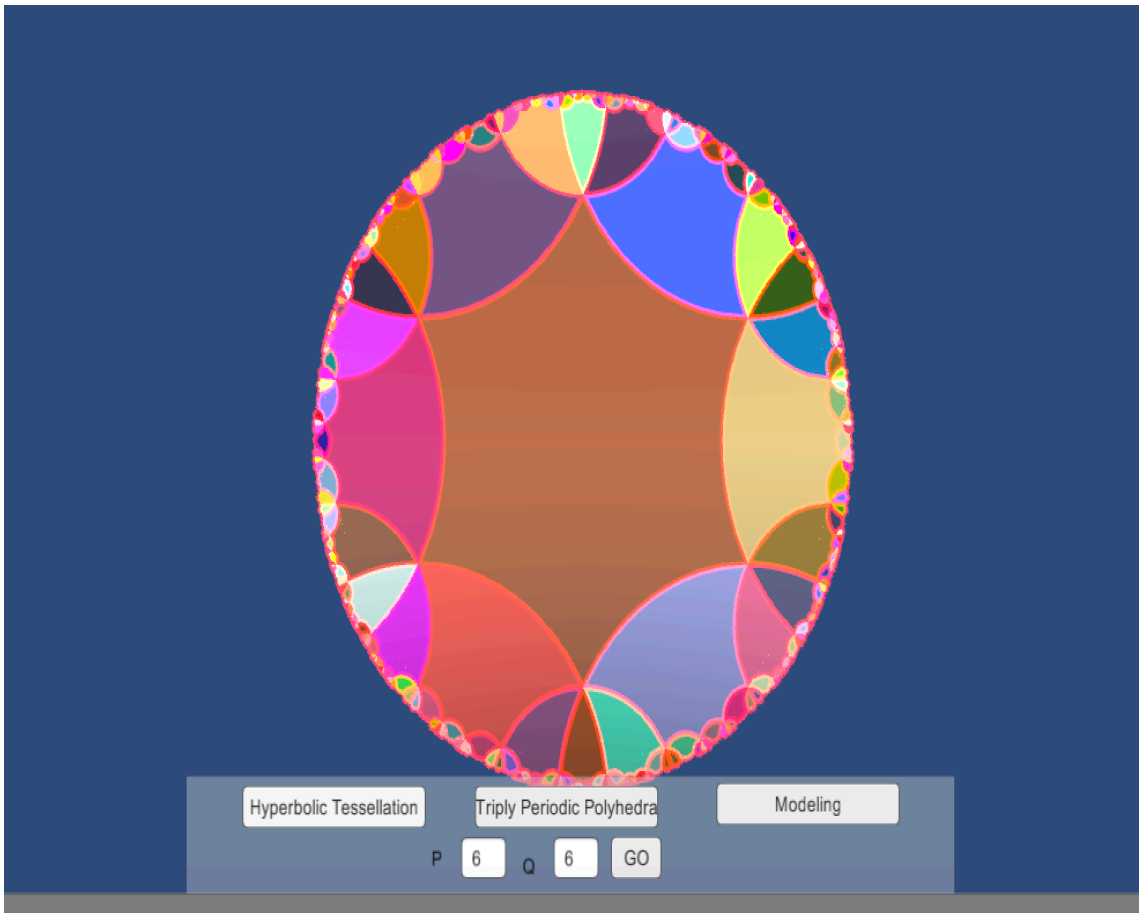

Figure 6.2: A {6,6} hyperbolic tessellation with 3 layers and randomly colored polygons

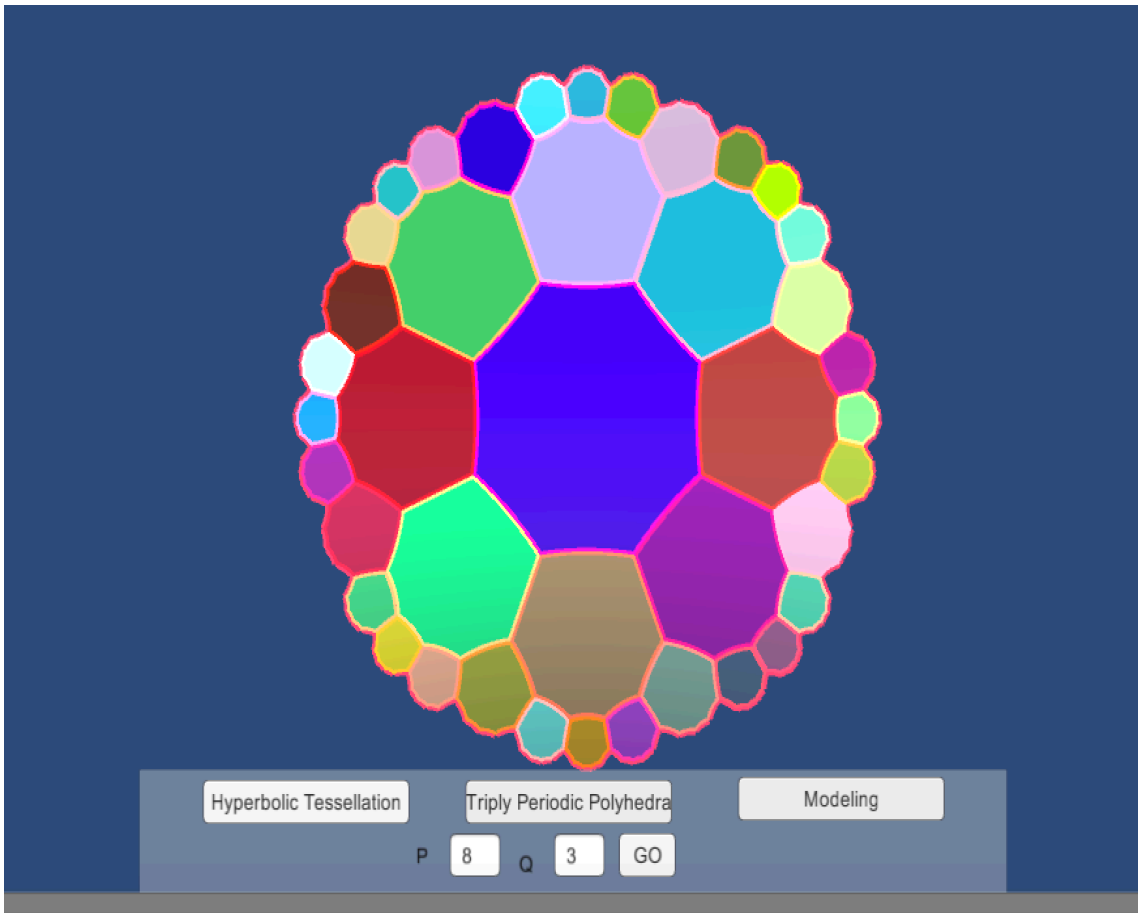

Figure 6.3: A {8,3} hyperbolic tessellation with 3 layers and randomly colored polygons

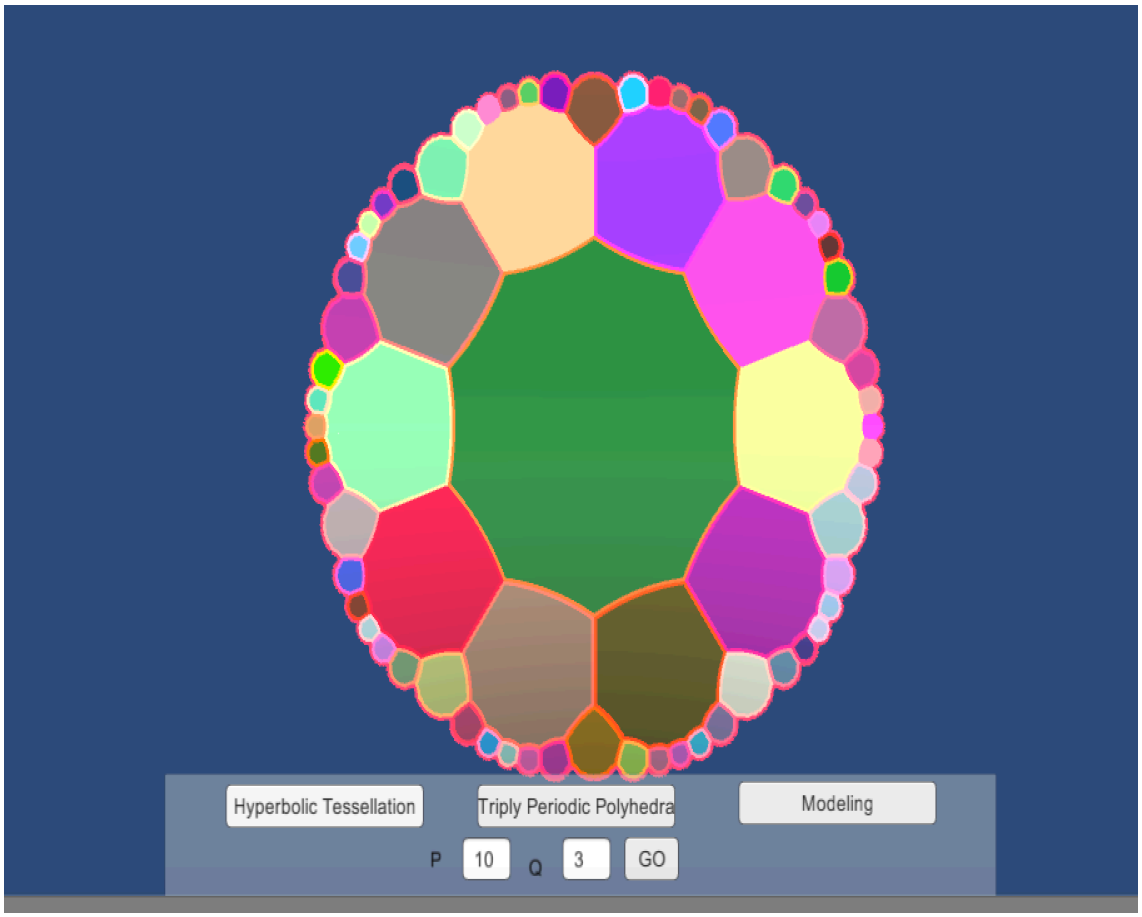

Figure 6.4: A {10,3} hyperbolic tessellation with 3 layers and randomly colored polygons

Figure 6.5: A {10,3} hyperbolic tessellation with 4 layers and randomly colored polygons

7 Conclusions

The main focus for this thesis was to create an application that allows the creation of hyperbolic tessellations and triply periodic polyhedrons. A few applications allow us to create hyperbolic tessellations, but are slightly complicated to operate. This application is very simple and intuitive to use. The application also has a section which models the relationship between a hyperbolic tessellation and a corresponding triply periodic polyhedron. This modelling is represented with simple and intuitive features. This application gives the user the ability to specify p and q to create the {p,q} hyperbolic tessellation. The modelling section also has the angels and demons patterns attached to both the hyperbolic tessellation and the corresponding triply periodic polyhedron.

The application can be extended to allow user to add the values of p,q and r and generate the {p,q|r} triply periodic polyhedron, provided the values are valid. The application can be extended to support more textures in addition to the angels and demons patterns. The application can be also extended to allow the creation of some texture which can then be applied to these hyperbolic tessellations and triply periodic polyhedrons. Also, the application can be extended to support the modelling with the help of more hyperbolic tessellations and corresponding triply periodic polyhedrons. Finally, since Unity has great support for viewing 3-D object using the Oculus rift, the triply periodic polyedrons can be viewed using the Oculus rift. The application can have features to add some textures inside the triply periodic polyhedrons, which can then be seen using the Oculus rift.

8 Bibliography

[1] K. Bhumkar. ``Interactive visualization of Hyperbolic geometry using the Weierstrass model". MA thesis. USA: Univeristy of Minnesota Duluth, 2006 (cit. on p. 4).

[2] S. K. Chittamuru. ``An Interactive Java Program to Generate Hyperbolic Repeating Patterns Based on Regular Tessellations Including Hyperbolic Cirlces and Horocycles". MA thesis. USA: Univeristy of Minnesota Duluth, 2015 (cit. on pp. 2, 10).

[3] H. Coxeter. ``The trigonometry of hyperbolic tessellations". In: Canad. Math. Bull. 40, 1997, pp. 158–168. URL: http://cms.math.ca/10.4153/CMB-1997-019-0 (cit. on p. 21).

[4] D. Dunham. ``Patterned Triply Periodic Polyhedra". In: 2012, pp. 275–282. URL: https://www.d.umn.edu/~ddunham/bridges12.pdf (cit. on pp. 1, 2, 29).

[5] D. Dunham, J. Lindgren, and D. Witte. ``Creating repeating hyperbolic patterns". In: *Proceedings of the 8th annual conference on Computer graphics and interactive techniques*. SIGGRAPH '81. New York, NY, USA: ACM, 1981, pp. 215–223. ISBN: 0-89791-045-1. DOI: 10.1145/965161.806808. URL: http://doi.acm.org/10.1145/965161.806808 (cit. on p. 33).

[6] M. J. Greenberg. *Euclidean and Non-Euclidean Geometries*. Second. New York: W.H. Freeman and Company, 1980 (cit. on pp. 5, 10, 13).

[7] M. Myers. *Tour of Tessellations*. URL: http://edtech2.boisestate.edu/meganhoopesmyers/502/virtualtour/history.html (cit. on p. 1).

[8] L. Olortegui. *Different Kinds of Geometry*. URL: http://www.ehow.com/info_8774739_different-kinds-geometry.html (cit. on p. 3).

[9] J. C. Polking. *Basic Information about spheres*. URL: http://math.rice.edu/~pcmi/sphere/sphere.html (cit. on p. 6).

[10] C. Selfstudier - Own work. URL: https://commons.wikimedia.org/w/index.php?curid=18419030 (cit. on p. 14).