UNIVERSITY OF MINNESOTA

This is to certify that I have examined this copy of a master's thesis by

Swetha Naidu

and have found that it is complete and satisfactory in all respects,
and that any and all revisions required by the final
examining committee have been made.

---

Name of Faculty Advisor

---

Signature of Faculty Advisor

---

Date

GRADUATE SCHOOL

# Transforming Euclidean Object files to Hyperbolic data files

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

Swetha Naidu

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

Dr Doughlas Dunham

June 2016

© Swetha Naidu 2016

Acknowledgements

I would like to take the opportunity to express my gratitude to all the people who motivated me towards the completion of the thesis.

Firstly, I hearfully thank Dr. Douglas Dunham for guiding me throughout the thesis. I was very much motivated to work with him. I enjoyed working with him. He has motivated me and helped me finish my thesis successfully.

I would specially thank Dr. Peter Willemson for his motivation and guidance. I would also thank Dr. Joseph Gallian for his support. I would like to thank Dr. Neda Saeedloei, Dr. Ted Peteren, Dr. Hudson Turner, Dr. Huayang Wang for teaching excellent courses and helping me strengthen computer sceince basics. I would also thank Clare Ford and Lori Lucia for providing support any time.

Finally, I would like to thank my parents and my brother for their timely support and encouragement.

# Dedication

I would like to take this opportunity to dedicate this to my mother, Mrs. Padmavathi Nuthalapati, my father Mr. Swamulu Naidu, my brother Mr. Karthik Gowtham.

Abstract

The creation of repeating hyperbolic patterns has been of interest for more than 100 years. However, manual creation of these patterns was very tedious and time-consuming. Escher created such patterns manually in 1950's. At that time, he had no access to computers. Later on, several researchers, tried to create these patterns automatically. For this purpose, they had written code in C. Programs for automatic creation of these patterns now exist in C++ and JAVA. These programs take parameters p and q to generate a regular tesselation p,q of hyperbolic plane. Here p represents a regular p-sided polygon and q specifies the number of them that meet at each vertex. These program generates repeating hyperbolic patterns from hyperbolic data files. However, there is no means to generate hyperbolic data files from Euclidean object files.

The main focus of this research is to generate hyperbolic data files from Euclidean object files. These hyperbolic data files are then given as input to a hyperbolic repeating pattern program. An algorithm has been implemented for the purpose of converting these Euclidean object files into hyperbolic data files. The goal of the research was to take an object file that describes a fractal euclidean pattern and produce a hyperbolic data file describing a corresponding fractal pattern. That hyperbolic data file describes a corresponding fractal pattern. That hyperbolic data file could then be used as input to an existing hyperbolic pattern program to create a repeating hyperbolic pattern with global symmetry, but which is locally fractal.

# Contents

# List of Figures

# 1 Introduction

The creation of repeating patterns has been done throughout history. Cultures such as Arabic, Chinese, Japanese, Moors, Persians, Romans have all used these tesselations. These are all Euclidean in nature.

Euclidean geometry was codified by Euclid in his "Elements." It illustrates the world around us by managing two dimensional and three dimensional objects.[4]

The dutch artist M.C.Escher became very much fascinated by Euclidean tilings[9]. He was inspired to create hyperbolic art after seeing a repeating hyperbolic pattern in an article by H.S.M. Coxeter. He then combined hyperbolic geometry and art in his four circle limit patterns.

Escher created these patterns manually since during that time, he did not have computer assistance. It was very tedious and time consuming process to produce such complex patterns.

Later several mathematicians tried to program the patterns that have been manually created by Escher. An algorithm was developed by Dr Dunham. It was implemented earlier in Fortran, Pascal, C,C++ and Java. This thesis focuses on providing input to that program written by Vejendla Maneesha[1] from Euclidean object files. The algorithm focuses on converting the Euclidean object files into hyperbolic data files. After that the generated

hyperbolic data files are given as input to that program and that program generates repeating hyperbolic patterns. A sample program output is shown in section 6.

# 2 Classical Geometry

Geometry means measurement of earth. It is the study of various different kinds of shapes and sizes. Geometry deals with these shapes in total. It is the core of everything existing. The topics that are usually covered are angles, intersecting lines, right triangles, perimeter, area, volume, circles, triangles etc. There are several different kinds of geometry such as Euclidean, hyperbolic and spherical geometries[2].

Geometry is a study of shapes, sizes, and positions that are within a given space. During the time of Euclid, there was no clear difference between physical space, geometrical space and other abstract shapes. Eventually, geometry was divided into various types that are described in the following section. There are two main types of geometry: Euclidean and non Euclidean.

Five postulates were stipulated by the Greeks. They built a huge body of knowledge from these axioms. All the geometric principles that are taught in school recently are the ideas of Greek mathematicians. On the whole, geometry is a means of measuring things in the world. It involves using the pictures to understand all the concepts clearly related to mathematics.[6]

## 2.1 Euclidean geometry

Euclidean geometry studies various kinds of plane and solid figures such as squares, triangles, circles etc. on the basis of theorems proposed by the Greek mathematician Euclid.

The book "Elements" written by Euclid was accepted as the textbook of geometry until recently. The concepts in this book are still taught in schools.

The axioms stated by Euclid which are the basis for Euclidean geometry are

- Two points can be joined to form a line.
- Any line segment with end-points given can be extended in either direction.
- A circle can be constructed with any given center and radius
- All right angles are congruent.
- Given a line l and a point p not on l, there is one and only one line which contains the point and is parallel. The axiom is known as "parallel postulate".

## 2.2 Hyperbolic geometry

Among the five postulates in Euclid's book "Elements", the fifth postulate is the most difficult one among them. If given the first four postulates, the fifth one is equivalent to

Playfair's axiom.[3]

Many researchers[7] felt that the parallel postulate was superflous and tried to deduce the fifth postulate from the first four postulates. They assumed that the postulate is false and tried hard to find a contradiction. However, they discovered many interesting results. Instead of obtaining a contradiction, they obtained a consistent geometry, which is called Hyperbolic Geometry.[8]

The hyperbolic axiom, which cannot be derived from the other four axioms, states that if there is a line R and a point P not on line R, there are at least two distinct lines A and B parallel to R, passing through P. see Figure 2.1

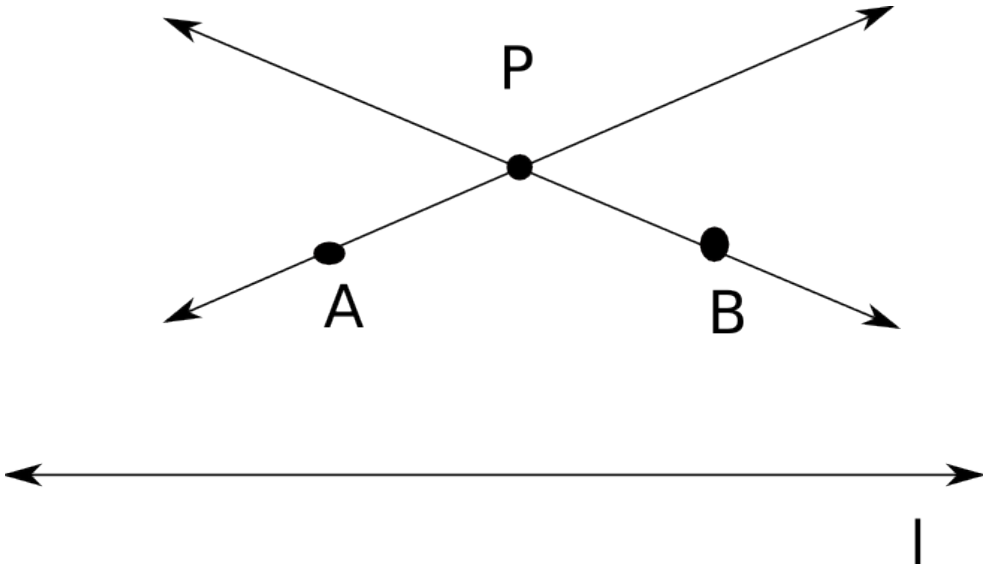


Figure 2.1: Lines in hyperbolic geometry

Based on this axiom, various theorems of hyperbolic geometry can be proved.

- All the angles in a triangle sum to a value less than 180.

- If two triangles have the same corresponding angles, they are congruent.

- There are no similar triangles that are not congruent.

- The angle sum of different triangles may not be the same.

## 2.3 Plane geometry

In geometry, a plane can be defined as a flat surface that is infinite in any direction. Usually, a coordinate plane is taken as an example. It has two axes x and y. An example of a plane is illustrated below in Figure 2.2. A plane can be defined as an infinite extension of a two-dimensional surface.

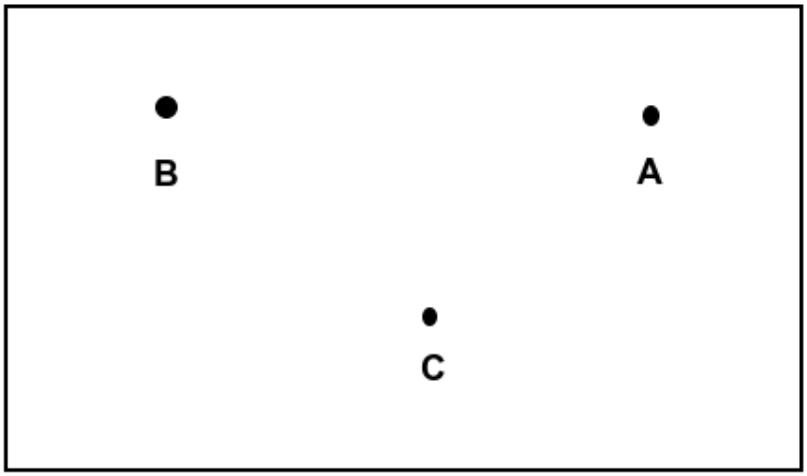


Figure 2.2: points in Euclidean plane

# 3 Hyperbolic Geometry Models

A model represents hyperbolic space and satisfies all the axioms of hyperbolic geometry. There are two types of models: finite and infinite. Finite models are used for representing the objects in Euclidean 2-D space whereas infinite models represent them in Euclidean 3-D space. The Poincare Disk Model and Klein Model are examples of finite models. The Weirstrass Model is an infinite model. An "Isomorphism" is a process of converting coordinates of one model into another. Isomorphisms exist between all the three models, Poincare Disk Model, Klein Model and Weirstrass Model.[3]

## 3.1 Poincare Disk Model

The Poincare Disk Model was invented by Jules Henri Poincare. He is the founder of algebraic topology and the theory of analytic functions. He has written several popular books on mathematics and essays on mathematical thinking and philosophy [10]. The Poincare Disk Model is a two-dimensional space which models hyperbolic geometry. A point in this model has the same meaning as in Euclidean Geometry. Every point (x,y) in this model can be represented as within a Euclidean bounding circle

$$x^2 + y^2 < 1 \tag{3.1}$$

A line can be represented in two ways. One way of representation is by open chords that pass through the center of the circle. A second way of representation is by any open

circular arc that is perpendicular to the bounding circle. The figure below illustrates this.

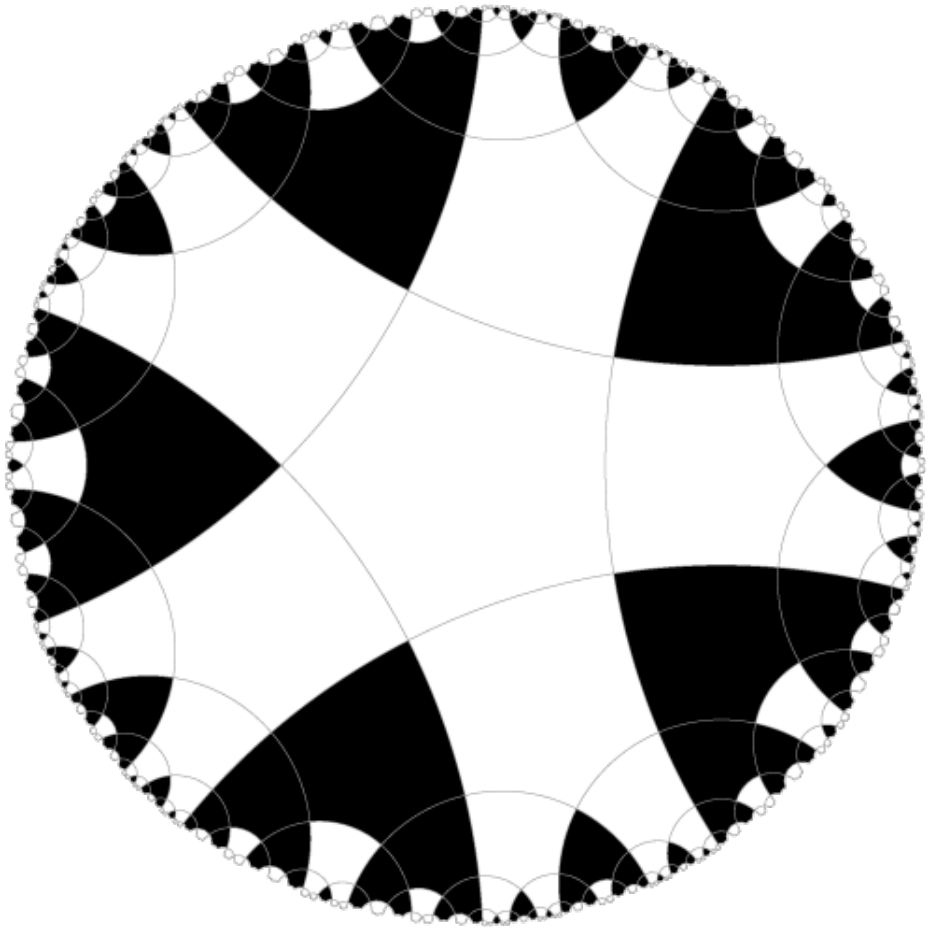

Figure 3.1: An image of the Poincare Disk

A line can be represented as an arc of a circle whose ends are perpendicular to the disk's boundary. The two arcs that do not meet correspond to parallel lines and arcs which meet orthogonally correspond to perpendicular lines. This model is conformal - angles have Euclidean measure.

## 3.2 Beltrami-Klein Model

The Beltrami-Klein model was proposed by a German mathematician Felix Klein. It is also known as Klein Model. It consists of an open disk in the Euclidean plane whose open chords represent the hyperbolic lines. There is much of similarity between this model and the Poincare Disk Model.

A point in this model is defined as any point inside a unit circle centred at the origin.

$$x^2 + y^2 < 1 \tag{3.2}$$

Two lines m and n, are said to be "parallel" if their chords fail to intersect and "perpendicular" in the cases:

1. If either m or n is a diameter of the disk, they are said to be hyperbolically perpendicular if and only if they are also perpendicular in Euclidean sense. 2. If neither of m nor n is a diameter, m is perpendicular to n if and only if the Euclidean line extending m does pass through n's pole. This model is illustrated in Figure 3.2.

## 3.3 Weierstrass Model

The Weierstrass Model is a representation of hyperbolic space on the hyperboloid's surface. A hyperboloid can be defined as a 3-dimensional surface which is cone-shaped. The main advantage of this model is that the other two models can be obtained by the projections from this model.[11]

The mathematical equation for this model is represented as

$$< X, X >= x^2 + y^2 - z^2 = -k^2 \tag{3.3}$$

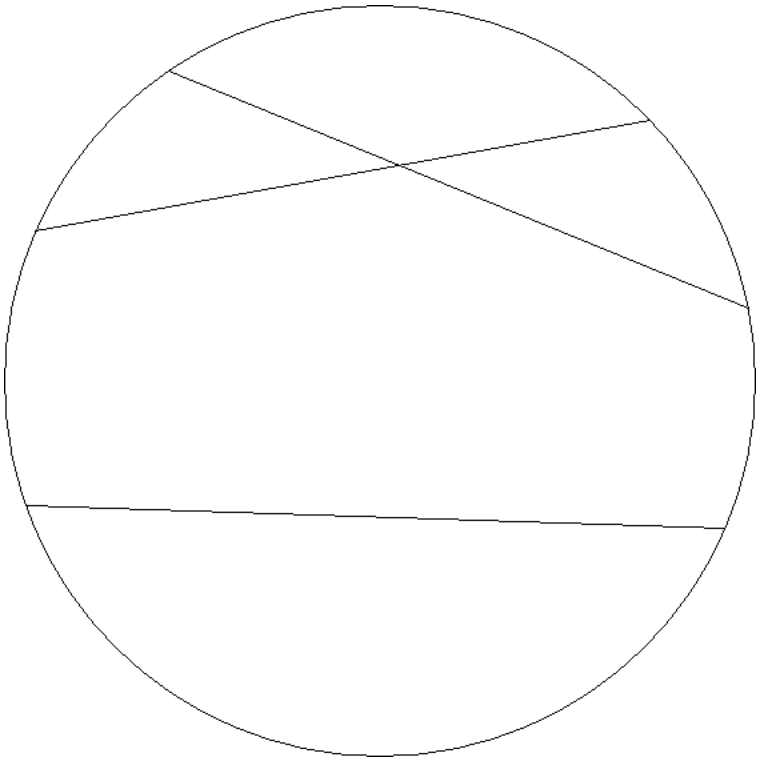

Figure 3.2: Beltrami Klein Model

where X is a vector (x,y,z). This gives two sheets: an upper sheet and a lower sheet. The points on lower sheet can be represented as reflections of the points on the upper sheet. So, we discard the lower sheet. If X1=(x1,y1,z1) and X2=(x2,y2,z2) then **<X1,X2>=x1x2+y1y2-z1z2**. A point in this model is represented using this equation. Any point that satisfies this equation lies on the hyperboloid. A line L can be obtained by intersecting the hyperboloid with a plane through the origin[3]. If a point X= (x,y,z) lies on line L, then

$$< X, L >= 0, z > 0 \tag{3.4}$$

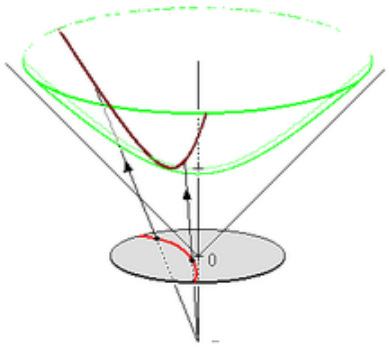

Figure 3.3: Weierstrass Model

## 3.4 Isomorphism

For any algorithm to work with two different models, it is desirable to be able to map objects from one model to other. Isomorphism is a process of mapping the objects from one model to another. Isomorphism preserves the properties and structures. There are different types of isomorphisms depending on the models.

### 3.4.1 Isomorphism between Weierstrass and Poincare Models

An isomorphism from Weierstrass to Poincare can be obtained by stereographic projection of Weierstrass Model onto XY plane to the point (0,0,-1). To map a point W[x,y,z] on the Weierstrass Model onto Poincare Disk Model, we use the below equation, [3]

$$[x, y, z] \rightarrow \frac{1}{z+1}[x, y, 0] \tag{3.5}$$

Similary, an isomorphism from the Poincare Disk model to the Weierstrass model can be defined as follows

$$[x, y, 0] \rightarrow \frac{1}{1-x^2-y^2}[2x, 2y, 1+x^2+y^2] \tag{3.6}$$

### 3.4.2 Isomorphism between Weierstrass and Klein Models

The projection of the Klein model on the z=1 plane to the point (0,0,0) given by

$$[x, y, z] \rightarrow [\frac{x}{z}, \frac{y}{z}, 1] \tag{3.7}$$

The Klein model to Weierstrass model inverse projection is given by

$$[x, y, 1] \rightarrow \frac{1}{1 - x^2 - y^2}[x, y, 1] \tag{3.8}$$

### 3.4.3 Isomorphism between Poincare and Klein Models

A point on the Poincare Disk Model is mapped to the Klein Model using the following equation.

$$x = \frac{2u}{1 + u.u + v.v} \tag{3.9}$$

$$y = \frac{2v}{1 + u.u + v.v} \tag{3.10}$$

For a vector k representing a point in the Klein Model, the inverse projection on the Poincaré Disk Model is given by the below equation:

$$u = \frac{x}{1 + \sqrt{1 - x.x - y.y}} \tag{3.11}$$

$$v = \frac{y}{1 + \sqrt{1 - x.x - y.y}} \tag{3.12}$$

# 4 Hyperbolic Patterns

So far, various kinds of geometries, and models have been discussed. In the first two sections below tessellations and patterns are explained. The third section discusses the symmetry operations used to generate these patterns. The fourth sections defines the motif and a fundamental region, the pattern generation algorithm, and the last section describes the implementation of the replication algorithm which is used in the pattern generation algorithm.

## 4.1 Tessellations

A tessellation can be defined as a repeating pattern that can be formed by translating the congruent copies of a basic sub-pattern. A tessellation is a repeated tiling of a surface with no overlaps and gaps.[3]

## 4.2 Repeating hyperbolic pattern

A repeating pattern is formed by a process of transformation and translation of a basic sub-pattern called a motif. A repeating hyperbolic pattern, is made of hyperbolically congruent copies of a motif. The hyperbolic patterns are "repeating patterns" from this point. An example is given in the Figure 4.1 below.[3]

A regular tessellation p,q on the hyperbolic plane. The notion p,q indicates that it has regular p-sided polygons and q of them meet at each vertex. We use p,q to denote such a

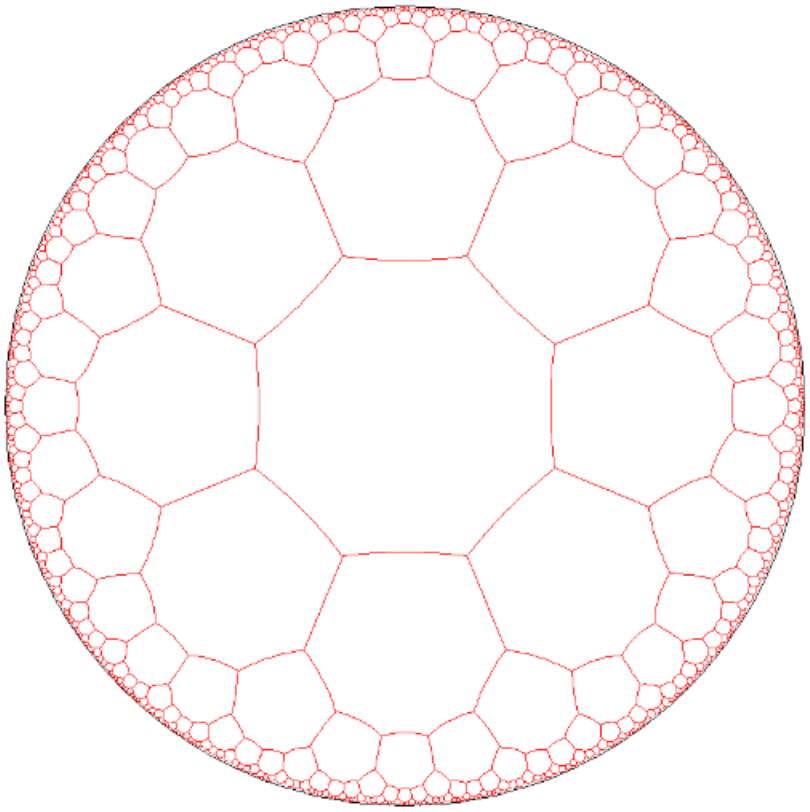

Figure 4.1: The regular tessellation 8,3 on a hyperbolic plane.

regular tessellation from this point.

For a tessellation p, q to be in the hyperbolic plane, the condition (p - 2) (q - 2) > 4 must be satisfied. When (p - 2) (q - 2) = 4, it is a Euclidean tessellation and when (p - 2) (q - 2) < 4, it is a spherical tessellation. Figure 4.1 shows the regular tessellation 8,3 [5].

## 4.3 Motif

A motif is a basic sub-pattern that is used for generating the repeating pattern. If the hyperbolic plane is covered by transformed copies of a connected set under elements of a symmetry group and if there are no overlaps, that set is called a "fundamental region". The final repeating hyperbolic pattern of fundamental regions is interlocking. A motif is

Figure 4.2: A computer generated version of Escher's Circle Limit II pattern based on 8,3 tessellation

contained with in a fundamental region and can fill out all of it. Figure 4.4 shows a motif for a fish pattern.

## 4.4 Repeating Pattern Generation Algorithm

This algorithm presented in the section was proposed by Dr.Dunham . This algorithm has been previously implemented by Vajendla[1]. My task was to provide the input data files to her program by converting Euclidean object files. The main process of replicating the motif for the purpose of generating the entire repeating pattern involves two steps. The first step in this process is to replicate the fundamental region and create the central p-gon pattern. The central p-gon pattern is created by the process of rotation of the motif

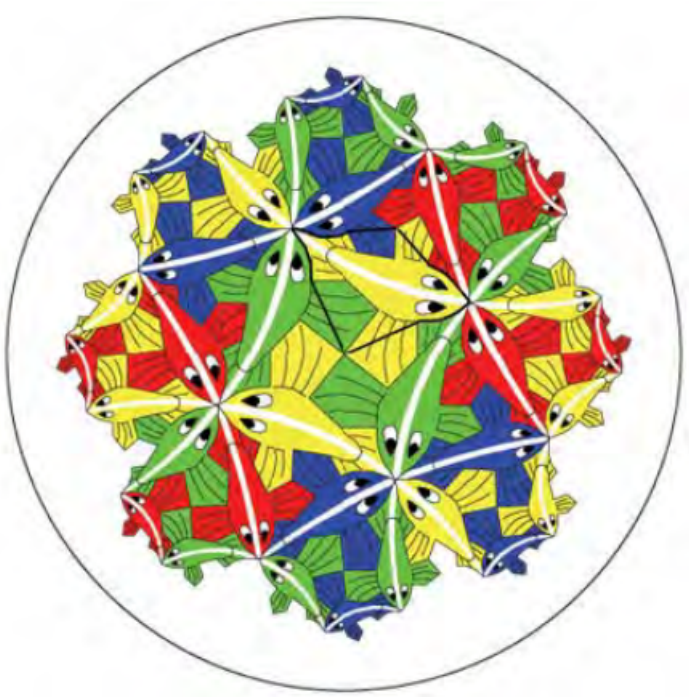

Figure 4.3: A sample motif outlined in black

around the p-gon center and/or reflecting across diameters and perpendicular bisectors of the edges until it has been completely filled. This central p-gon pattern forms the first layer of repeating pattern.

The second step in this process is the replication of the p-gon pattern in the form of layers. It is more efficient to replicate the entire p-gon pattern than replicating the motif because the number of transformations can be greatly reduced and these transformations are less prone to roundoff errors. These layers are generated recursively. The first layer in this pattern is generated using the first step of the algorithm and the k+1 layer consists of all the p-gons sharing an edge or vertex with the k-th layer. The layer generation to 4 layers is illustrated in the figure 4.6 The p-gon 1 is rotated about the vertex A to draw p-gon 2, which is rotated about vertex B to draw p-gons 3, which is again rotated about vertex C to draw p-gons 4.

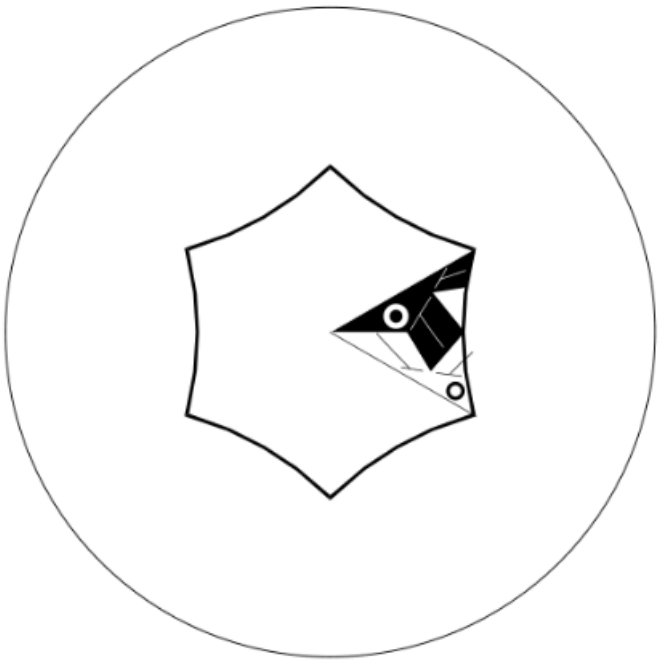

Figure 4.4: Pattern with central p-gon before replicating the motif

All the points of the fundamental region have undergone symmetry operations to transform the p-gon from one layer to the next layer. Every point is projected onto the Weierstrass Model by using the inverse projected formula. By taking the product of the vector representing the coordinates with the Lorenz matrix representing the symmetry operation, the point is then transformed to a new location. The new point is then projected back to the Poincare Model [3].

## 4.5 Implementing the Replication Algorithm

To generate the (k+1)th layer from the kth layer, the algorithm iterates over each vertex of the p-gon in the kth layer which it shares with (k+1)th layer. For each vertex, the algorithm calculates the number of polygons that are needed to draw the next layer from that vertex. Then the algorithm calls itself recursively for the vertices of the p-gons in the newly

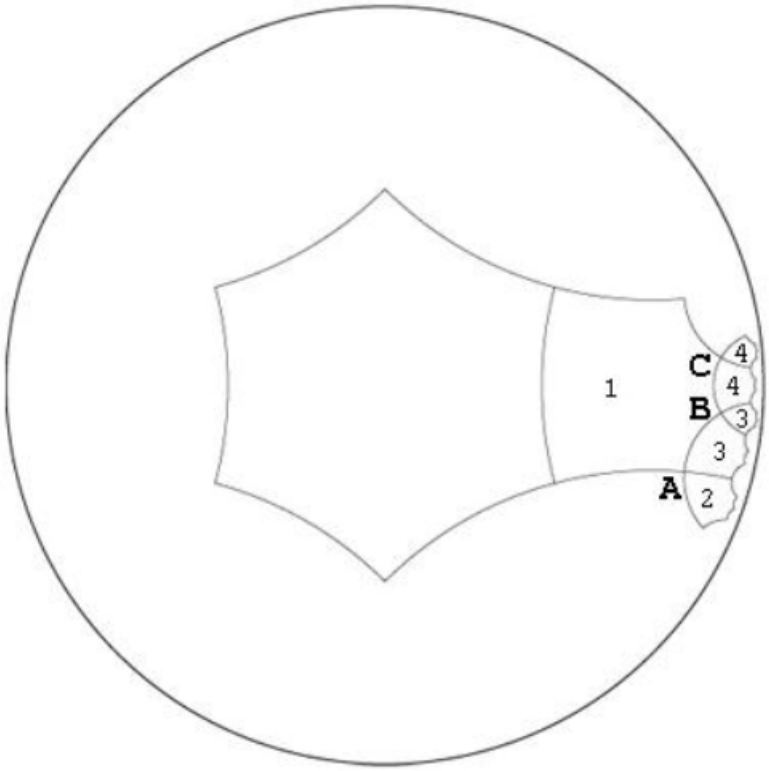


Figure 4.5: Pattern with central p-gon after replicating motif

formed layer. This process continues by calling the algorithm for all the exposed vertices of the new p-gons.

# 5 Implementation

This chapter focuses on the algorithm that converts the Euclidean object files into hyperbolic data files which are given as input to generate the repeating hyperbolic patterns. The Euclidean object file represents one quarter of the circle to be used in a hyperbolic 4-gon, the other three quarters are obtained by 90 degree rotations.

## 5.1 Algorithm to convert the Euclidean object files into hyperbolic data files

The steps that are involved in the algorithm are stated below. In the example below, the Euclidean objects are circles.

- Step 1: Start with an Euclidean object file. A Euclidean object (of circles) consists of lines of four numbers; x and y coordinates of the center of the circle, the radius r of the circle and a code 0 or 1 indicating whether that circle has been duplicated by a 90 degree rotation about origin.

  This step invloves reading the values of x, y, and r as an arraylist of Euclidean objects. While reading the data, care is taken not to duplicate points. This is indicated by the fourth number in a line of input data. If the fourth number in the input data is zero it indicates that the point is duplicate and the condition is specified to discard the duplicate point. These Euclidean data files are stored into an arraylist of Euclidean

object files. So the arraylist contains only unique circles and all the duplicates are eliminated.

• Step 2: The next step is to decide on the parameters p and q. These parameters are set to values of 4 and 6, respectively. These are determined by the Euclidean object file. A future extension would allow for other values of p and q.

• Step 3: Compute the corners in the Poincare Model. The purpose of computing the corners in a Poincare model is accomplished by the calculation of the coordinates of xqpt and yqtpt which represent the coordinates of the Poincare Model.

• Step 4: Transfer the corners in Poincare model back to Klein Model. The Poincare model coordinates are transformed into Klein model. Two formulas are used for this purpose. These are described in great detail in the next section.

• Step 5: Scale. Compute the scale from Klein coordinates of the corners.

• Step 6: Apply a scale to a Euclidean data file coordinates to put them into the Klein Model.

• Step 7: Apply the Klein to Poincare mapping to transfer the Euclidean circle from the Klein model.

- Step 8: Repeating Hyperbolic Patterns are generated by applying this input to Vajendla's program.

## 5.2 Reading the input file

One practice when reading the input file is to eliminate any kind of duplicate records that can be present in the input file. The duplicate record can be identified by a value of zero given to the fourth column in the input file. If the fourth column reads zero it implies that it is a duplicate record and while reading the input file, that record is eliminated from reading. Thus the input arraylist consists of only unique records and duplicate records are eliminated. All the input is stored in the form of ArrayList and the arraylist consists of a collection of Euclidean points. The Euclidean points are formed by variables of a circle and a number. The circle is formed by variables of x coordinate, y coordinate and radius. Thus all the input rows are read in the form of euclidean points and they are placed in the ArrayList. Thus the input ArrayList is a collection of Euclidean points. The parameters for p and q are selected as 4 and 6, respectively.

Initially a circle is constructed by taking the three input numbers and converting them to doubles. For each point that is in the input file, four different points are placed in the ArrayList, each rotated 90 degrees from the previous one. After constructing the four different points, Euclidean points are constructed from the four different circles. The Euclidean points are placed in the input ArrayList. In this way, all the points are stored in an ArrayList.

## 5.3 Calculating the Klein coordinates

For the purpose of calculating the Klein coordinates, initially the coordinates of xqpt and yqpt are calculated, when xqpt and yqpt are the coordinates of the vertex of the central 4-gon in the first quadrant. After computing the coordinates of xqpt and yqpt, they are converted into Klein coordinates by following the formulae listed below. After conversion, the Klein object is returned. Let u and v be the initial x and y coordinates, and let the converted klein coordinates be x and y. The formulae for conversion is given below.

$$x = \frac{2 * u}{1 + u * u + v * v} \tag{5.1}$$

$$y = \frac{2 * v}{1 + u * u + v * v} \tag{5.2}$$

And the Klein coordinates are returned to the main program.

## 5.4 Applying scale to the Klein coordinates

In order to scale the Klein coordinates, first a scaleFactor is calculated as the x coordinate of Klein divided by 10.0, since all the coordinates of the Euclidean input files have been scaled up by a factor of 10.0. After that, for every Euclidean input file, Klein to Poincare conversion is applied by taking the Euclidean input, the ScaleFactor and the Klein coordinates into account.

The procedure that is followed for KtoP conversion is given below.

Initially, the coordinates of xNew and yNew are calculated. The formula for calculating the coordinates of xNew and yNew is given as the x coordinate of Klein divided by 10

multiplied by the x coordinate of the euclidean circle.

$$xNew = \frac{klein.xk}{10} * e.circle.x \tag{5.3}$$

Similarly, the procedure for calculating the yNew is given as the y coordinate of the Klein corner divided by 10 multiplied by the y coordinate of the euclidean circle.

$$yNew = \frac{klein.yk}{10} * e.circle.y \tag{5.4}$$

The procedure for calculating the rNew is given as the x coordinate of the corner divided by 10 multiplied by the radius of the Euclidean circle.

$$rNew = \frac{klein.xk}{10} * e.circle.radius \tag{5.5}$$

After that, the value of R is calculated.

The formula for calculating the value of R is given as below

$$R = \sqrt{xNew * xNew + yNew * yNew} \tag{5.6}$$

After calculating the value of R, the value of cos and sin are calculated by the following formulae

$$cos\theta = \frac{xNew}{R} \tag{5.7}$$

$$sin\theta = \frac{yNew}{R} \tag{5.8}$$

Let x1, y1, x2, y2 be the new coordinates.

Their values are calculated by the following formulae

$$x1 = xNew + rNew * cos\theta \tag{5.9}$$

$$y1 = yNew + rNew * sin\theta \tag{5.10}$$

$$x2 = xNew - rNew * cos\theta \tag{5.11}$$

$$y2 = yNew - rNew * sin\theta \tag{5.12}$$

Then, the values of x1P, y1P, x2P, y2P are calculated from them by applying the Klein to Poincare transformation. The formulae are listed below.

$$x1P = \frac{x1}{1 + \sqrt{1 - x1 * x1 - y1 * y1}} \tag{5.13}$$

$$y1P = \frac{y1}{1 + \sqrt{1 - x1 * x1 - y1 * y1}} \tag{5.14}$$

$$x2P = \frac{x2}{1 + \sqrt{1 - x2 * x2 - y2 * y2}} \tag{5.15}$$

$$y2P = \frac{y2}{1 + \sqrt{1 - x2 * x2 - y2 * y2}} \tag{5.16}$$

After that the value of xCent, yCent, xEdge, yEdge are calculated by the following formulae

$$xCent = \frac{x1P + x2P}{2} \tag{5.17}$$

$$yCent = \frac{y1P + y2P}{2} \tag{5.18}$$

$$xEdge = x1P \tag{5.19}$$

$$yEdge = y1P \tag{5.20}$$

After calculating these coordinates, the coordinates are wrriten to the output file as two points.

The points have their coordinates as xCent, yCent, 1, 3, 6 and xEdge, yEdge, 1, 3, 6 where the integers 1, 3, 6 are used by the hyperbolic program where 1 represents color black, 3 represents circle, and 6 is not used.

These points are written to the output file.

There is a special case for the origin point. If the point is origin, then the following condition is satisfied, then the following method is used. useKtoP(rNew,0.0).

The useKtoP() method is used for converting the x,y coordinates into u and v. The formula that is used for this purpose is given as

$$u = \frac{x}{1 + \sqrt{1 - x * x - y * y}} \tag{5.21}$$

$$v = \frac{y}{1 + \sqrt{1 - x * x - y * y}} \tag{5.22}$$

After this conversion, the following points are written into the output file,

0, 0, 1, 3, 6 u, v, 1, 3, 6

Since, while reading the input file, four different coordinates are read for each input

point, the output file contains eight different points for each input point. It also contains two different points for each origin point. Thus, the output file contains eight points for each input point and two points for the origin coordinate. Finally, the output file is given as input to Maneesha Vajendla's program to generate a repeating hyperbolic patterns. The screen shots of the output files are presentes in the results section.

# 6 Results

This chapter provides the results that have been generated from the data files that have been generated from the conversion of the Euclidean data file to a hyperbolic data file. After converting into the hyperbolic data file, the data file is given as input to Maneesha's repeating pattern generation program. It generates the repeating hyperbolic data file as output. That program is written in Java. The main aim of that program is to convert the Euclidean data file into repeating data pattern. It has a nice GUI to take the input from the user and display the results. It has several options to support various shapes such as

1. circle,
2. filled polygon,
3. polyline,
4. hyperbolic line segment,
6. hyperbolic line,
7. equidistant curve,
8. horocycle.

The results are also displayed as shown below.

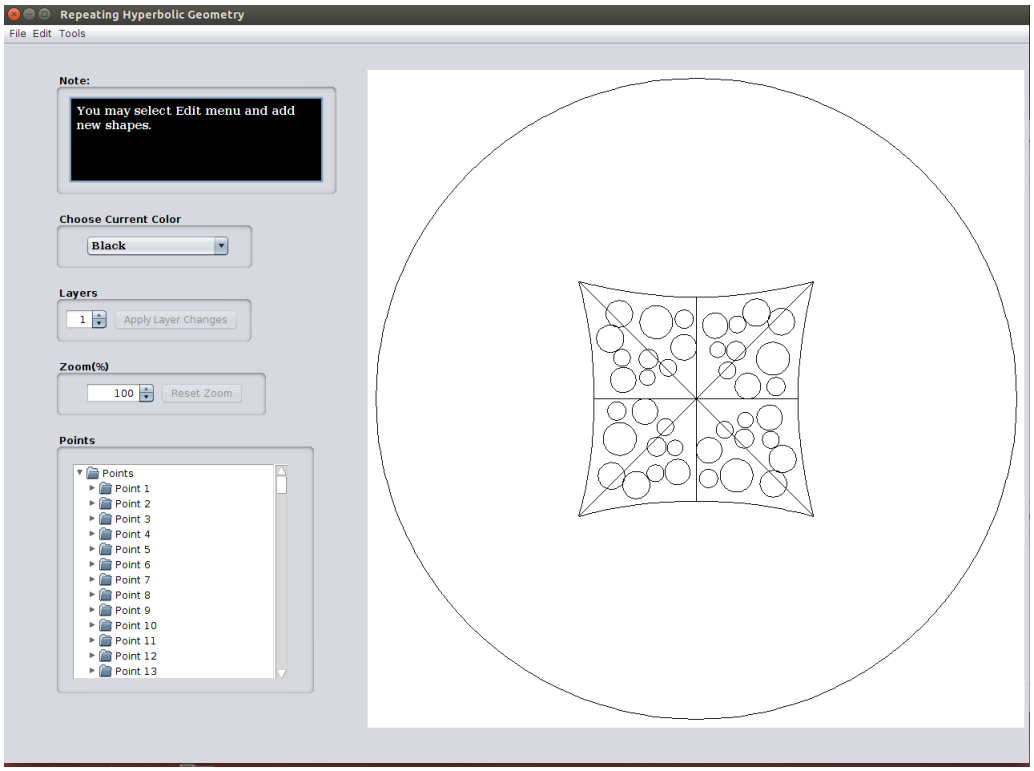


Figure 6.1: The repeating pattern that has been generated by converting the first 10 points in the .dat file

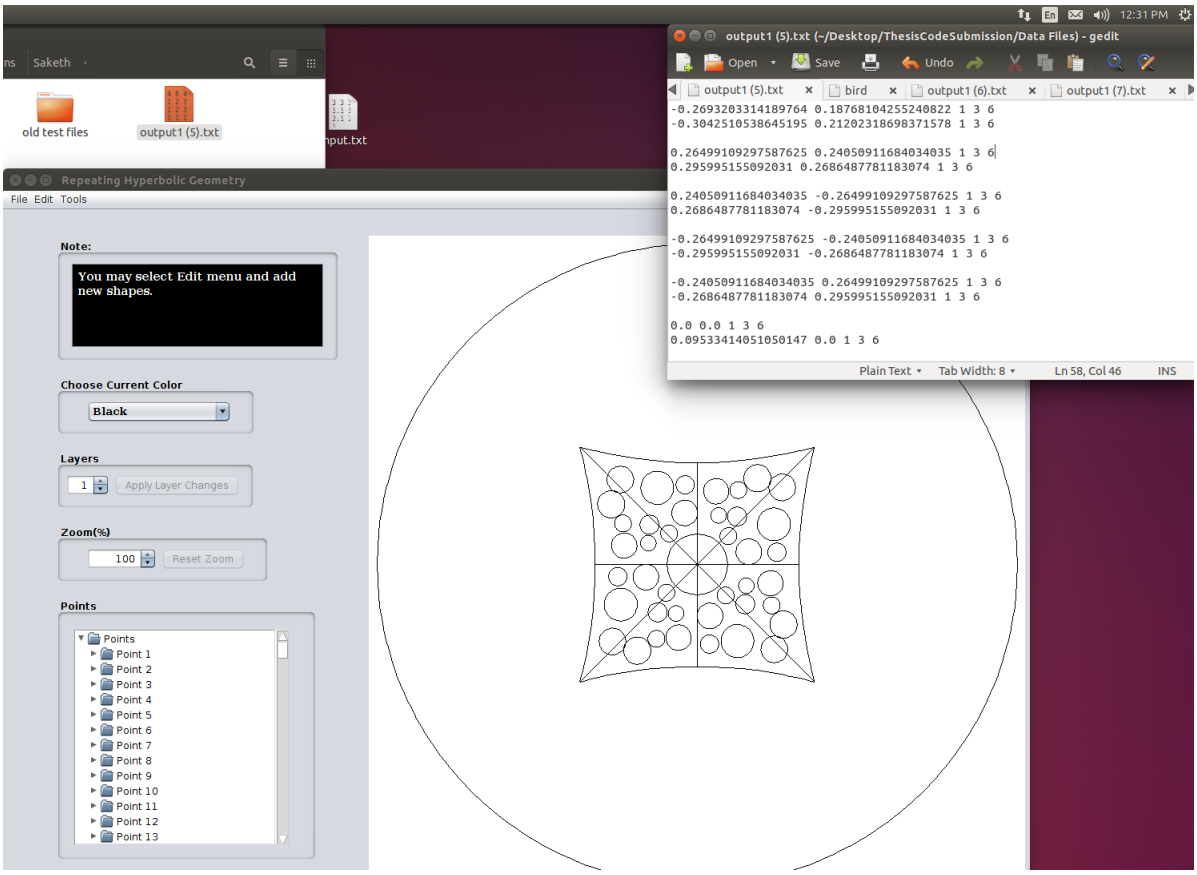


Figure 6.2: The repeating pattern that has been generated by converting the first 10 points in the .dat file along with origin

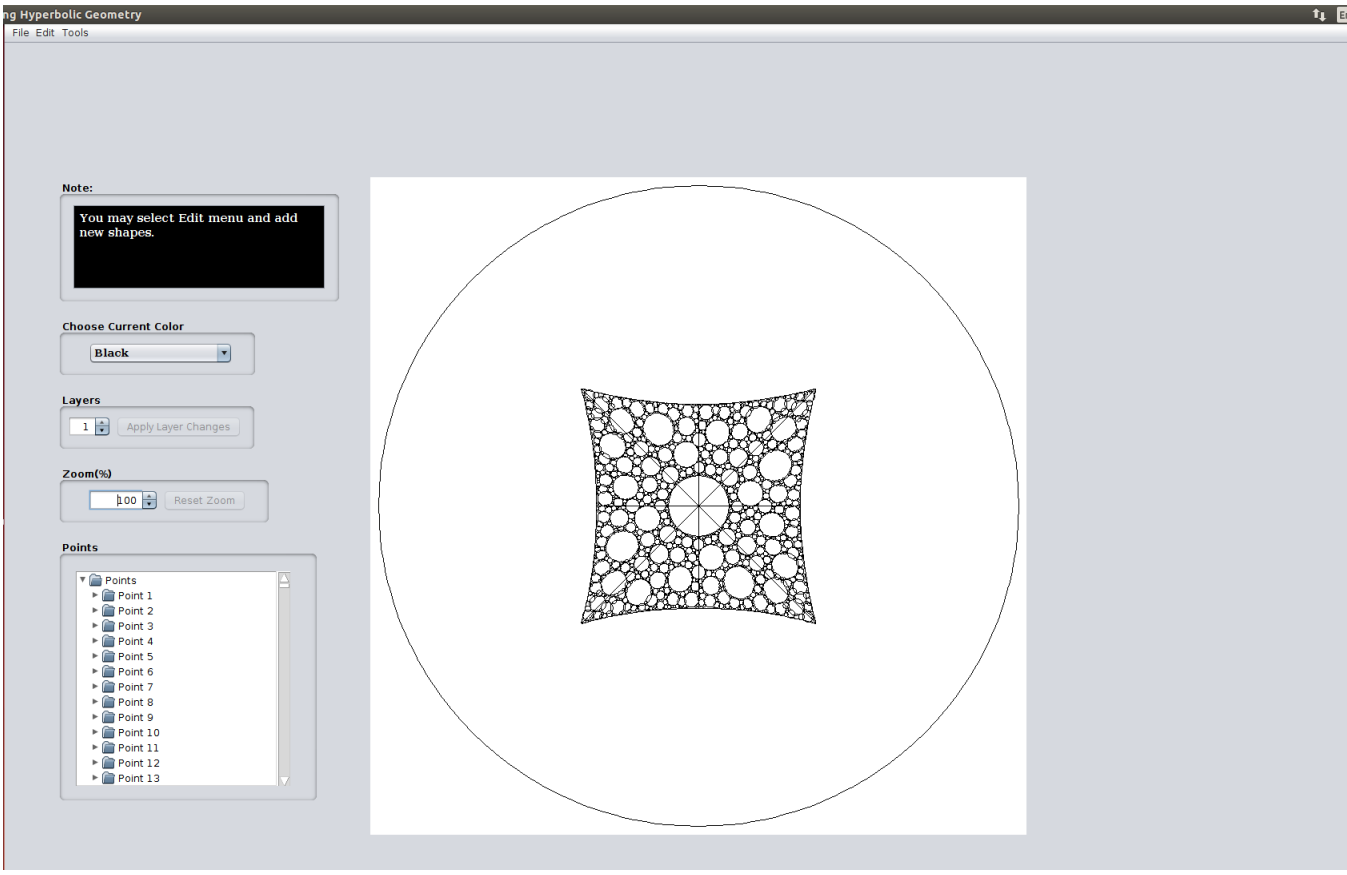


Figure 6.3: The repeating pattern that has been generated by converting the first 50 points in the .dat file along with origin

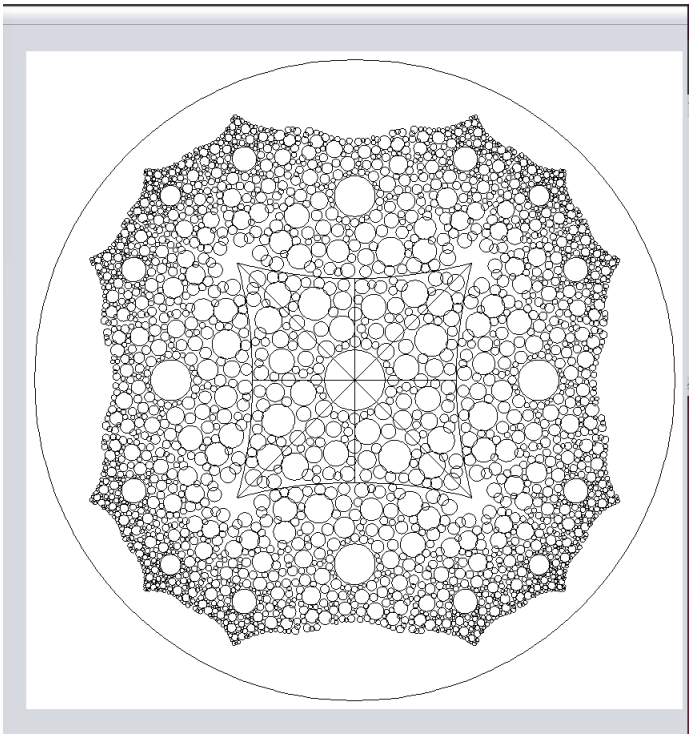

Figure 6.4: The repeating pattern that has been generated by converting 50 points in the .dat file and showing 2 layers

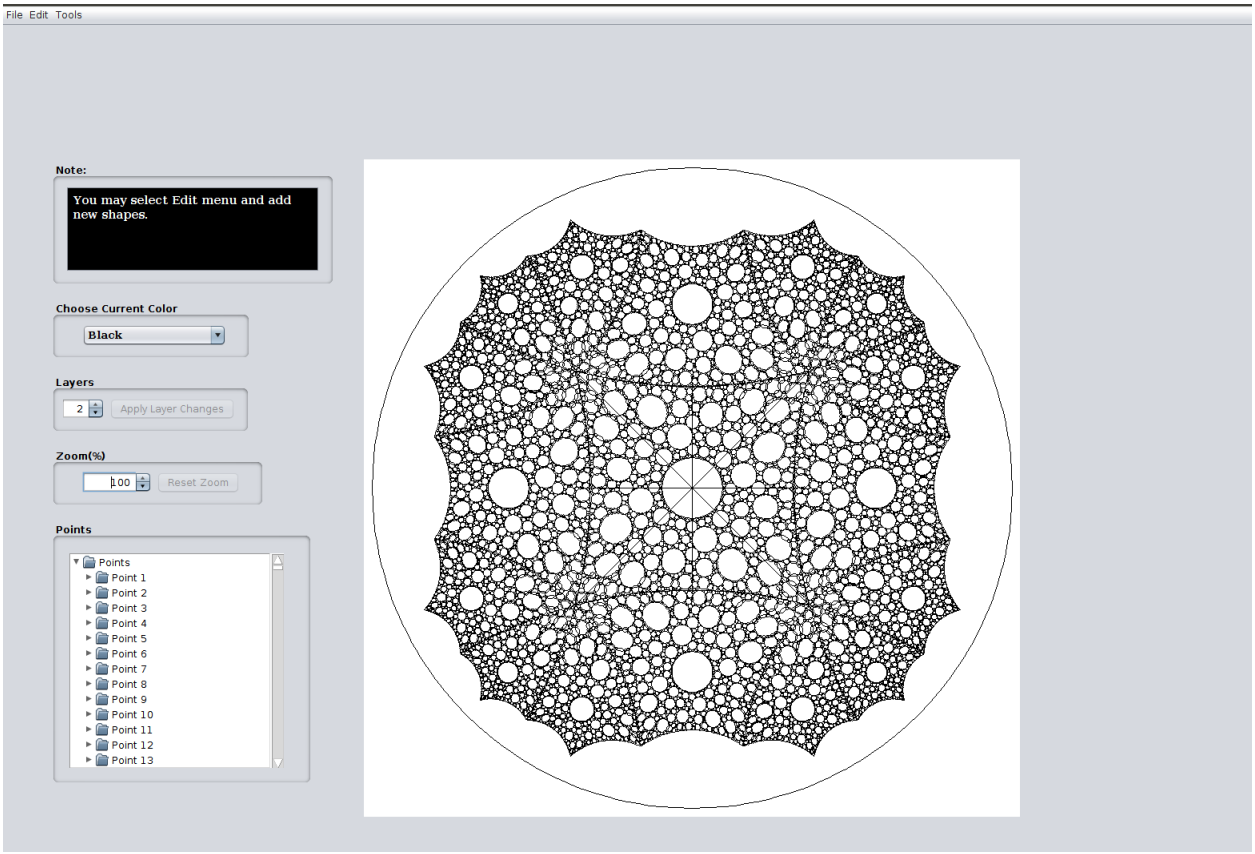


Figure 6.5: The motif of the repeating pattern that has been generated by converting all the points in the .dat file and showing 2 layers

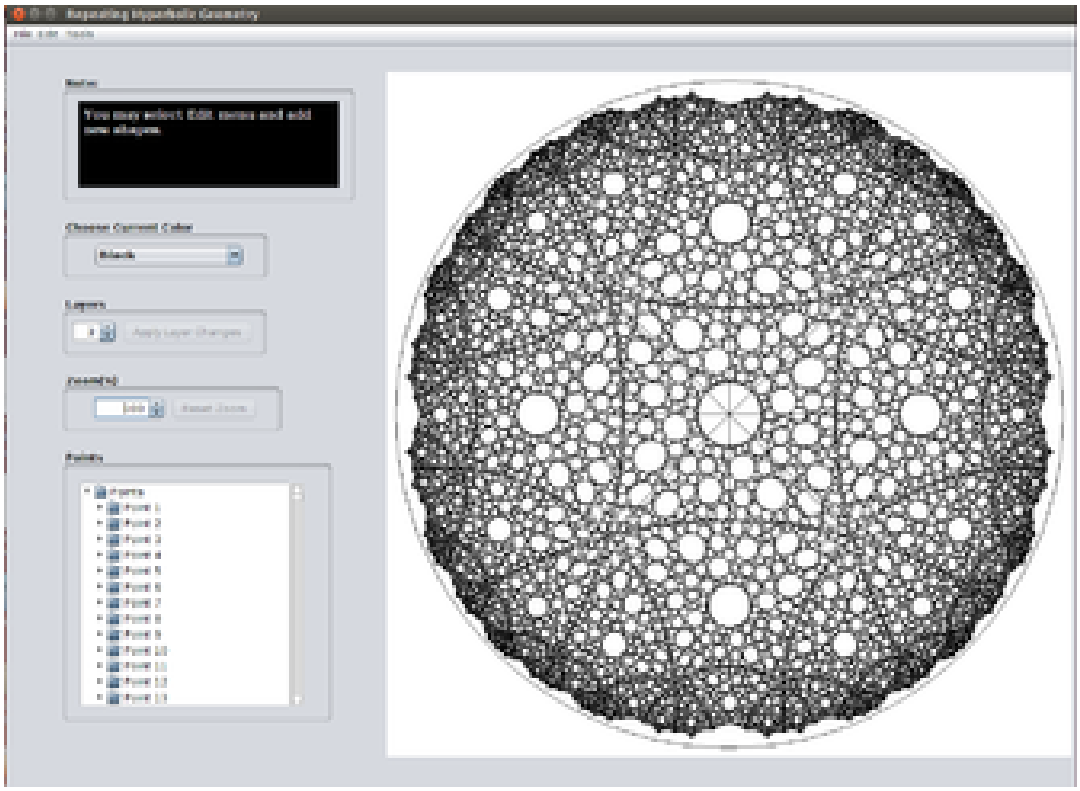

Figure 6.6: The motif of the repeating pattern that has been generated by converting all the points in the .dat file and showing 3 layers

# 7 Conclusions

This research focused on the conversion of a Euclidean data file into hyperbolic data file based on parameters p and q. This task has been accomplished through a program which performs that conversion. The program was written in Java and is thus portable to many platforms. As a result of this work, hyperbolic data files have been generated and given as input to Vajendla's program which succesfully generated repeating hyperbolic patterns. These patterns are locally fractal in nature, as was the pattern of the Euclidean input file, but they are globally repeating with [4,6]+ symmetry.

# Bibliography


[1] *An interactive Java program to generate Hyperbolic repeating patterns based on regular tessellations including hyperbolic lines and equidistant curves*. URL: http://d.umn.edu/cs/degrees/grad/documents/SakethKarumuriThesisJuly2015.pdf/ (cit. on pp. 1, 16).

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

[3] D. Dunham. ``Artistic patterns in hyperbolic geometry". In: *In Bridges 1999 Conference Proceedings, pages 239-249,1999()* () (cit. on pp. 5, 8, 11, 12, 14, 18).

[4] D. Dunham. ``D. 1986, Hyperbolic symmetry". In: *Computers and Mathemeatics with Applications, Volume 12B, pages 139-153. ()* () (cit. on p. 1).

[5] *Generating Repeating patterns based on regular tessellations using an Applet*. URL: http://www.d.umn.edu/cs/thesis/Vejendla_Maneesha_December2013.pdf (cit. on p. 15).

[6] *Geometry*. URL: http://www.coolmath.com/reference/geometry-trigonometry-reference (cit. on p. 3).

[7] *Hyperbolic Geometry*. URL: https://faculty.etsu.edu/gardnerr/noneuclidean/hyperbolic.pdf (cit. on p. 5).

[8] *Hyperbolic Geometry*. URL: https://en.wikipedia.org/wiki/Hyperbolic_geometry (cit. on p. 5).

[9] A. Parsekhar. ``A Unified data representation and Visulatization of Patterns based on regular tessellations in the three classical geometries. Master's Thesis, University of Minnesota Duluth 2002". In: () (cit. on p. 1).

[10] *Poincare Disk Model*. URL: http://www.mathworld.wolfram.com/PoincareHyperbolicDisk.html (cit. on p. 8).

[11] *Weierstrass Model*. URL: http://www.mathworld.wolfram.com/WeierstrassFunction.html (cit. on p. 10).

# A Appendix A

Data File format

This section explains the format of the input data file for this program. Here is the input data file "p4.dat" that has been used for generating the hyperbolic data files. The first 50 lines of the input file are given below.

w 10.000000 h 10.000000 414

0.000000 0.000000 3.272727 1 0

7.668691 4.026864 1.435057 1 1

5.395155 1.323778 1.284303 1 1

1.916035 7.463528 1.166577 1 1

5.844005 8.386086 1.071725 1 1

8.105302 7.356470 0.993432 1 2

4.130378 4.971838 0.927546 1 2

3.289521 2.989634 0.871220 1 10

8.078465 1.236629 0.822430 1 6

2.234064 5.146722 0.779697 1 40

4.133117 7.502229 0.741912 1 6

5.312334 3.515890 0.708226 1 17

5.808630 5.870275 0.677977 1 8

0.037259 6.864064 0.650642 1 1

6.864064 -0.037260 0.650642 0 1

0.726804 9.313948 0.625800 1 12

9.252260 5.909231 0.603110 1 2

1.240340 3.847207 0.582291 1 25

0.038582 4.090429 0.563111 1 86

4.090429 -0.038582 0.563111 0 86

9.103409 2.497222 0.545374 1 46

2.553519 9.221470 0.528915 1 18

9.162634 8.826470 0.513596 1 48

9.441336 1.105446 0.499295 1 131

7.519327 9.401196 0.485910 1 7

7.024861 2.229832 0.473352 1 95

4.556368 9.223671 0.461542 1 34

3.572206 6.396884 0.450413 1 23

0.685396 5.418354 0.439904 1 54

9.488422 3.501961 0.429962 1 74

0.341530 8.186501 0.420541 1 27

8.186501 -0.341531 0.420541 0 27

6.743892 6.971890 0.411598 1 70

7.297995 6.127983 0.403097 1 150

9.508059 7.561181 0.395003 1 49

9.452312 4.739376 0.387287 1 199

3.711240 1.005466 0.379922 1 50

8.384067 9.304319 0.372882 1 47

2.201305 3.724528 0.366147 1 2

1.070081 6.162553 0.359694 1 37

3.781060 8.853927 0.353507 1 17

4.578802 6.304199 0.347569 1 74

1.696877 9.474673 0.341863 1 106

3.383240 1.675326 0.336376 1 10

8.152767 5.784567 0.331095 1 34

5.425952 6.868896 0.326008 1 82

9.634734 6.752702 0.321103 1 364

3.744437 9.680975 0.316371 1 7

1.012736 4.736107 0.311803 1 308

# B Appendix B

The output data file that has been generated from this file is shown below

4 6 4 0 7 0

1 2 3 4 5 6 7

1 2 3 4 5 6 7

1 1 2 3 4 5 6 7

2 1 2 3 4 5 6 7

3 1 2 3 4 5 6 7

4 1 2 3 4 5 6 7

3202

0.0 0.0 1 3 6

0.09533414051050147 0.0 1 3 6

0.23913437976652183 0.125570534142546 1 3 6

0.284745604486866 0.14952119258246277 1 3 6

0.125570534142546 -0.2391343797665218 1 3 6

0.14952119258246277 -0.284745604486866 1 3 6

-0.23913437976652183 -0.125570534142546 1 3 6

-0.284745604486866 -0.14952119258246277 1 3 6

-0.125570534142546 0.2391343797665218 1 3 6

-0.14952119258246277 0.284745604486866 1 3 6

0.16074424323067113 0.03944088590882215 1 3 6

0.19987279666182048 0.049041632913121386 1 3 6

0.03944088590882215 -0.16074424323067113 1 3 6

0.049041632913121386 -0.19987279666182048 1 3 6

-0.16074424323067113 -0.03944088590882215 1 3 6

-0.19987279666182048 -0.049041632913121386 1 3 6

-0.03944088590882215 0.16074424323067113 1 3 6

-0.049041632913121386 0.19987279666182048 1 3 6

0.05863087595132644 0.22838475514656653 1 3 6

0.06852093972012344 0.2669095043605432 1 3 6

0.22838475514656653 -0.05863087595132644 1 3 6

0.2669095043605432 -0.06852093972012344 1 3 6

-0.05863087595132644 -0.22838475514656653 1 3 6

-0.06852093972012344 -0.2669095043605432 1 3 6

-0.22838475514656653 0.05863087595132644 1 3 6

-0.2669095043605432 0.06852093972012344 1 3 6

0.18768104255240822 0.2693203314189764 1 3 6

0.21202318698371578 0.3042510538645195 1 3 6

0.2693203314189764 -0.18768104255240822 1 3 6

0.3042510538645195 -0.21202318698371578 1 3 6

-0.18768104255240822 -0.2693203314189764 1 3 6

-0.21202318698371578 -0.3042510538645195 1 3 6

-0.2693203314189764 0.18768104255240822 1 3 6

-0.3042510538645195 0.21202318698371578 1 3 6

0.26499109297587625 0.24050911684034035 1 3 6

0.295995155092031 0.2686487781183074 1 3 6

0.24050911684034035 -0.26499109297587625 1 3 6

0.2686487781183074 -0.295995155092031 1 3 6

-0.26499109297587625 -0.24050911684034035 1 3 6

-0.295995155092031 -0.2686487781183074 1 3 6

-0.24050911684034035 0.26499109297587625 1 3 6

-0.2686487781183074 0.295995155092031 1 3 6

0.12403213326426868 0.14930054183524002 1 3 6

0.14318999779843536 0.17236133648643714 1 3 6

0.14930054183524002 -0.12403213326426868 1 3 6

0.17236133648643714 -0.14318999779843536 1 3 6

-0.12403213326426868 -0.14930054183524002 1 3 6

-0.14318999779843536 -0.17236133648643714 1 3 6

-0.14930054183524002 0.12403213326426868 1 3 6

-0.17236133648643714 0.14318999779843536 1 3 6

# C Appendix C

Step 1: Reading the input file

```
public static void readInputFile() throws IOException
{
   FileReader filereader=new
       FileReader("C:\\Users\\Swetha\\Desktop\textbackslash{}thesisfiles\textbackslash
   BufferedReader bufferedreader=new BufferedReader(filereader);
   String line=bufferedreader.readLine();
          line=bufferedreader.readLine();
   String[] split=new String[5];
   while(line!=null)
   {
      int length=line.length();\newline
      if(length==0)\newline
         break;\newline
      split=line.split(" +");\newline
      if(Integer.parseInt(split[3])!=0)\newline
      {
                      boolean x=Double.parseDouble(split[0])==0.0;
                      boolean y=Double.parseDouble(split[1])==0.0;
                  if(x\&\&y)
                  {
                     Circle circ1=new
```

```
            Circle(Double.parseDouble(split[0]),Double.parseDouble(split[1
            Double.parseDouble(split[2]));
         Euclidean euc1=new
            Euclidean(circ1,Integer.parseInt(split[3]));
         input.add(euc1);
      }
      else
      {
Circle circ1=new
   Circle(Double.parseDouble(split[0]),Double.parseDouble(split[1]),
   Double.parseDouble(split[2]));
Circle circ2=new
   Circle(Double.parseDouble(split[1]),-Double.parseDouble(split[0]),Dou
Circle circ3=new
   Circle(-Double.parseDouble(split[0]),-Double.parseDouble(split[1]),Do
Circle circ4=new
   Circle(-Double.parseDouble(split[1]),Double.parseDouble(split[0]),Dou
Euclidean euc1=new
   Euclidean(circ1,Integer.parseInt(split[3]));
Euclidean euc2=new
   Euclidean(circ2,Integer.parseInt(split[3]));
Euclidean euc3=new
   Euclidean(circ3,Integer.parseInt(split[3]));
Euclidean euc4=new
   Euclidean(circ4,Integer.parseInt(split[3]));
input.add(euc1);
input.add(euc2);
```

```
                input.add(euc3);
                input.add(euc4);
                }
          }
                     line=bufferedreader.readLine();
     }
   }
```

Step 4: Transfer the corners in Poincare model back to Klein Model

```
public static Klein PtoK()
{
     //(u,v)--poin-care-to-klein--(x,y)
       double u=3.660254037844388e-01;
       double v=3.660254037844388e-01;
       double x=2*u/(1+u*u+v*v);
      double y=2*v/(1+u*u+v*v);
       return new Klein(x,y);
}
```

Step 5: Calculate the scaleFactor

```
     scaleFactor=klein.xk/10.00
```

Step 6: Apply scale to Euclidean data file coordinates to put them in Klein model

```
  public static void scale(Klein klein) throws IOException
      {
         double scaleFactor=klein.xk/10.00;
         for(Euclidean e:input)
```

```
        {
            KtoP(e,scaleFactor,klein);
        }
    }
```

Step 7: Apply Klein to Poincare Mapping to transfer the Euclidean circle from Klein model

```
public static void KtoP(Euclidean e,double scaleFactor, Klein klein)
    throws IOException
{
        double xNew,yNew,rNew;
        double u1,v1;
        xNew=klein.xk/10*e.circle.x;
        yNew=klein.yk/10*e.circle.y;
        rNew=klein.xk/10*e.circle.radius;
        BufferedWriter bw = null;
    try
    {
    // APPEND MODE SET HERE
            bw = new BufferedWriter(new
                FileWriter("C:\\Users\\Swetha\\Desktop\\thesisfiles\\output.txt",
                true));
            bw.write(Double.toString(xNew)+" "+Double.toString(yNew)+"
                "+Double.toString(rNew)+"
                "+Double.toString(e.circle.radius)+" "+e.number);
            bw.newLine();
            bw.flush();
```

```
}
catch (IOException ioe)
{
       ioe.printStackTrace();
}
finally
{                    // always close the file
       if (bw != null)
       try
       {
          bw.close();
       } catch (IOException ioe2) {
   // just ignore it
}
}

if(e.circle.x*e.circle.x+e.circle.y*e.circle.y<0.000001)
{
    useKtoP(rNew,0.0);
}
else
{
   double R=Math.sqrt(xNew*xNew+yNew*yNew);
   double cosO=xNew/R,sinO=yNew/R;
   double x1,y1,x2,y2;
   x1=xNew+rNew*cosO;
   y1=yNew+rNew*sinO;
```

```
  x2=xNew-rNew*cos0;
  y2=yNew-rNew*sin0;
  double x1P=x1/(1+Math.sqrt(1-x1*x1-y1*y1));
double y1P=y1/(1+Math.sqrt(1-x1*x1-y1*y1));
  double x2P=x2/(1+Math.sqrt(1-x2*x2-y2*y2));
  double y2P=y2/(1+Math.sqrt(1-x2*x2-y2*y2));
  double xCent=(x1P+x2P)/2;
  double yCent=(y1P+y2P)/2;
  double xEdge=x1P;
  double yEdge=y1P;
   try
  {
  // APPEND MODE SET HERE
  bw = new BufferedWriter(new
      FileWriter("C:\\Users\\Swetha\\Desktop\\thesisfiles\\output1.txt",
      true));
  bw.write(Double.toString(xCent)+" "+Double.toString(yCent)+"
      "+new String("1")+" "+new String("3")+" "+new String("6"));
  bw.newLine();
  bw.write(Double.toString(xEdge)+" "+Double.toString(yEdge)+"
      "+new String("1")+" "+new String("3")+" "+new String("6"));
  bw.newLine();
  bw.flush();
  }
  catch (IOException ioe)
  {
  ioe.printStackTrace();
```

```
            }
            finally
            {                       // always close the file
            if (bw != null)
            try
            {
               bw.close();
            } catch (IOException ioe2) {
               // just ignore it
            }
         }
        }
}
```