Generating Repeating Hyperbolic Patterns based on Regular Tessellations using an Applet

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

Maneesha Vejendla

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

Dr. Douglas Dunham

December 2013

Acknowledgments

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

Firstly, I sincerely thank Dr. Douglas Dunham for his guidance throughout the course of my thesis. I enjoyed working with him. My special thanks to the former graduate student of UMD Christopher Becker who paved a way for my thesis work and being helpful in all phases of my work.

I would also like to thank Dr. Pete Willemsen and Dr. Marshall Hampton who filled energy and enthusiasm in us during their courses and for being a part of my thesis committee. I thank Dr. Hudson Turner, Dr. Ted Pedersen, Dr. Richard Maclin and Dr. Steven Trogdon for the great knowledge they shared. I would also thank Lori Lucia and Clare Ford and all CS department for their timely help.

Finally I thank my friends and fellow graduate students Kiran Kumar Bushireddy and Praveen Reddy Katta and my parents for their constant support and motivation.

Dedicated to
my mother
Mrs. Rama Devi Mupparaju,
my father
Mr. Muni Ratnam Vejendla
and my brother
Prudhvi Raj Vejendla

Abstract

Some of the hyperbolic patterns of the Dutch artist M. C. Escher, which are considered as the finest works of hyperbolic geometry art, are computer-generated using algorithms that create hyperbolic patterns by replicating a basic sub-pattern called the motif. Escher created his patterns by hand - a very tedious and time consuming task. This paper describes the creation of these patterns using a computer program. The current algorithms that generate these repeating patterns are based on the regular tessellation of the hyperbolic plane, {p, q}, where “p” denotes a regular p-sided polygon, and “q” specifies the number of them that meet at each vertex.

The focus of this research is to replicate these patterns using a Java applet, which makes the program portable across the platforms. The applet loads a data file that contains the information to generate a repeating pattern for the external user. The user can also create and modify such a data file using the user interface of the applet. The patterns generated are displayed on the screen quickly and precisely. The research uses Weierstrass Model of Hyperbolic Geometry for all calculations and Poincaré’s model of Hyperbolic Geometry as the basis for representations of the desired patterns.

Contents

List of Figures

Chapter 1

Introduction

The art of creating repeating patterns has had importance in various cultures of human history. Arabic, Byzantine, Chinese, Egyptians, Greek, Japanese, Moors, Persians, and Romans have all incorporated these tessellations in their artwork [1]. During the times of these civilizations people have replicated repeating patterns on floors, walls, ceilings, and buildings and in ceramics, fabrics, rugs, wallpaper, and stained-glass windows. Several great artists such as M. C. Escher, Victor Vasarely, and Bridget Riley have explored these patterns, designed them and represented them in various geometries such as the Euclidean plane, Hyperbolic plane, and the Sphere[2]. Escher's Circle Limit patterns are the examples of combining hyperbolic geometry with art [3].

All the work done by Escher to represent these patterns and designs was done by hand which was very tedious and time consuming. Escher applied transformations to his patterns to obtain new patterns, thus changing the symmetry of the original pattern, sometimes even forcing it onto entirely different geometry. This laid the foundation for other artists to create patterns with interlocking people and animals joined to form tilings on the plane. This is the motivation for this thesis.

Currently, there are few programs that create repeating hyperbolic patterns. Among them, the program given by Dr. Dunham allows for the transformation of a motif from symmetry group to another in hyperbolic geometry. Dr. Dunham's

program uses Poincaré and Weierstrass coordinates [4]. In order to computerize these designs and to represent these hyperbolic patterns based on regular tessellations, Christopher D. Becker provided a program written in C++ and using the Qt graphics library which creates the repeating hyperbolic patterns in less tedious and more timely manner [5]. However, it must be downloaded and compiled. The current work focuses on creating a JAVA Applet so that the program can be run directly by external researchers while also maintaining the backward compatibility of the data files (discussed in Appendix A).

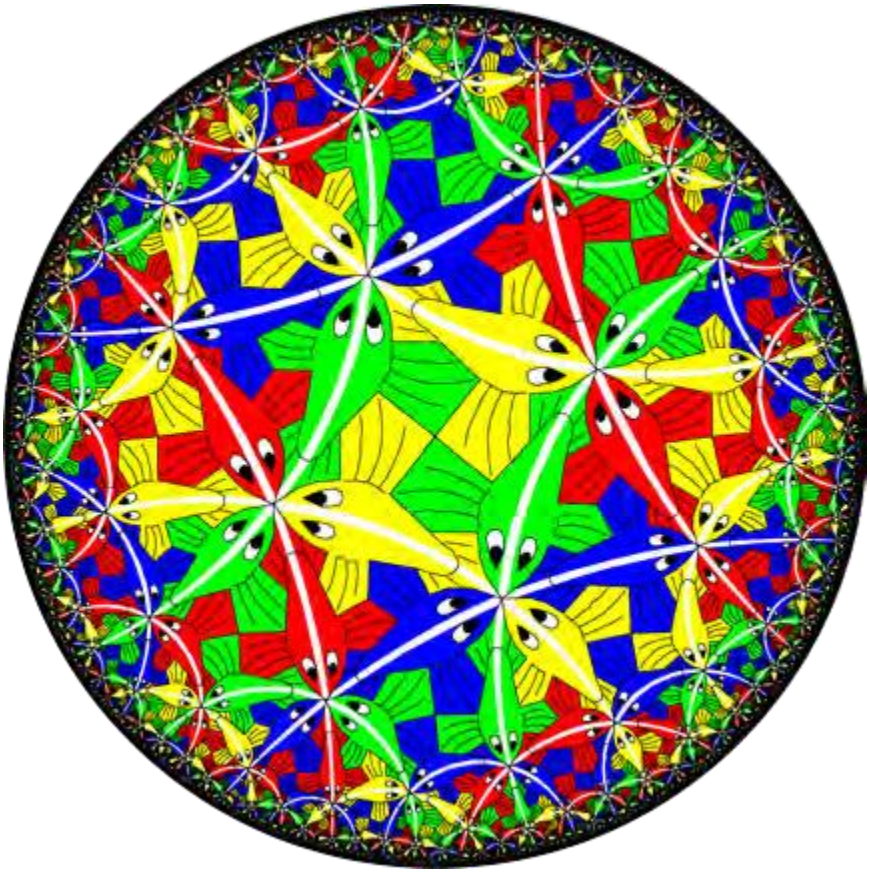

Figure 1.1: An example of program output

Chapter 2

Classical Geometries

Geometry is a branch of Mathematics that deals with study of shapes and sizes of objects of various dimensions. Geometry has been progressing enormously since the ancient times. Modern geometry deals with more complex problems such as the study of differential geometry and gravitational fields [6]. Geometric concepts are generalized to a high level of abstraction and complexity, and have been subjected to the methods of calculus and abstract algebra. Different kinds of geometry used for artistic purposes include Euclidean Geometry, Hyperbolic Geometry, and Spherical Geometry.

2.1 Euclidean Geometry

Euclidean geometry is the most familiar Geometry which was named after Euclid, a Greek mathematician from 300 BC. “The Elements” is a book by Euclid which contains axioms, theorems and proofs about basic geometrical figures such as squares, triangles, circles, etc. This book is 2000 years old, but the concepts from this book are still taught in high schools today.

Euclidean Geometry is also called an Axiomatic system, in which all the theorems were derived from axioms. “Euclid’s five axioms” are the base for all the theorems of Euclidean Geometry. All these 5 axioms are listed here:

1. A straight line can be drawn by joining two points.

2. Any line segment with given endpoints may be extended in either direction.

3. A circle can be constructed with any point as its center and with the radius of any length.

4. All right angles are equal to one another.

5. Given a line *l* and a point not on *l*, there is one and only one line which contains the point, and is parallel to *l*. This axiom is also known as the parallel postulate.

2.2 Hyperbolic Geometry

After Euclid proposed his fifth postulate, for over 2000 years many mathematicians believed that the parallel axiom was not needed. They tried to prove that it could be derived from the first four axioms. Three men, Ivanovitch Lobachevski from Russia, Karl Gauss from Germany, and János Bolyai from Hungary who were working separately in the early nineteenth century tried to develop theorems using Euclid's first four axioms and negation of the Parallel axiom. They wanted to prove that their negation of the Parallel axiom is inconsistent with the first four axioms. But to their surprise, they never obtained the contradiction, instead they developed a complete and consistent geometry, non-Euclidean that is now called Hyperbolic Geometry.

This proved that the parallel axiom can't be derived from the other four axioms. Another axiom called *Hyperbolic axiom* is used instead of this parallel postulate. This axiom states that if there exists a line *R* and a point P not on the line *R*, there

are at least two distinct lines (x and y) parallel to R passing through P (as shown in Figure 2.1).

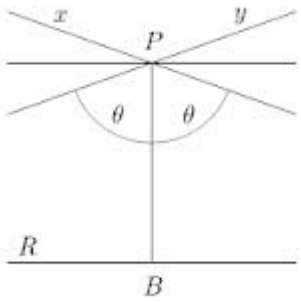

Figure 2.1 Hyperbolic Axiom

This axiom is used to prove the theorems of Hyperbolic geometry. Some of these properties are:

1. The sum of angles of a triangle is less than 180°.
2. Triangles with the same angles are congruent.
3. There are no similar triangles.
4. The sum of all angles is not same for all triangles [7].

There are various models to represent hyperbolic geometry. Some of these models like Poincaré Disk, Klein and Weierstrass models are discussed in the next chapter.

2.3 Spherical Geometry

Spherical Geometry is one of the most useful non-Euclidean geometries. Bernard Riemann and Ludwig Schläfli laid the foundations for this geometry. This is a 2-dimensional geometry on the surface of a sphere. Its lines are *great circles* on the surface of a sphere. A great circle is the largest circle that can be drawn on a sphere. These lines are defined as the shortest distance between two given points. For example longitudinal lines and the equator of the earth are the great circles whereas latitudinal lines are not. They divide sphere into two equal halves called as hemispheres.

There are various applications in which this spherical geometry is highly used. Some of them are spherical geometry is used in aircraft and ships for navigating around the globe. This is an interesting type of geometry involving some non-intuitive results. For example, it is surprising to know that the shortest flying distance from Florida to Philippine Island is a path across Alaska. And the reason is that Florida, Alaska, and the Philippines lie on the same great circle and so are collinear in spherical geometry. Another odd property of spherical geometry is that **the sum of the angles of a triangle is always greater than 180°** [8].

The spherical geometry parallel axiom is inconsistent with Euclid's first four axioms. This geometry also violates the Euclid's parallel axiom. An example representing the spherical geometry is shown in Figure 2.2

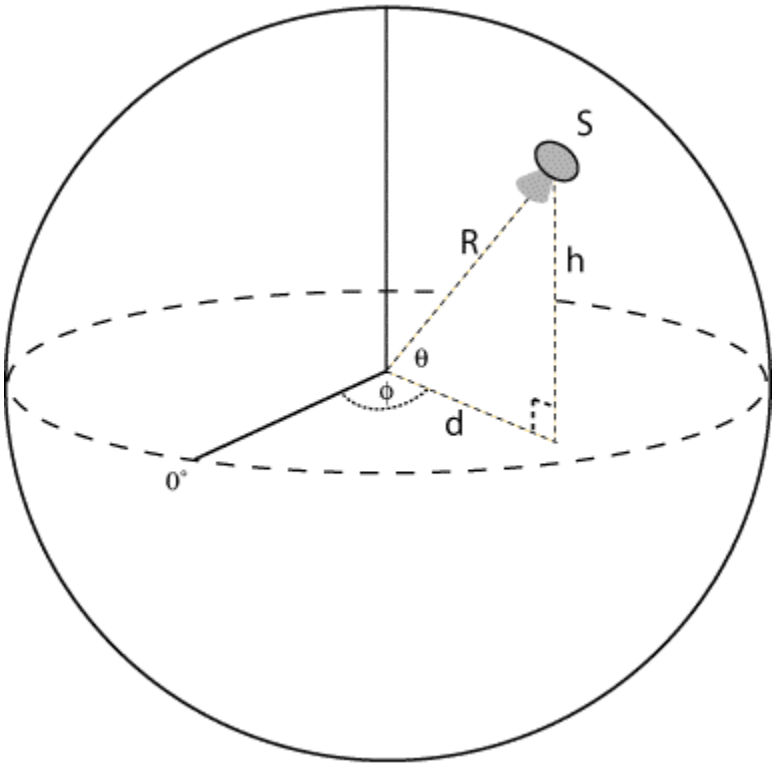

Figure 2.2 Representing spherical geometry

Chapter 3

Hyperbolic Geometry Models

A *model* is used to assign meanings to objects that make all the axioms in a given system true. Hyperbolic objects placed in positions in Euclidean 2D space can be represented using these models. The Poincaré disk model and the Beltrami-Klein model are the finite hyperbolic geometry models, and the Weierstrass model is an infinite model for hyperbolic geometry that is embedded in Euclidean 3-space. The isomorphism between the finite and infinite models are described in the following sessions.

3.1 Hyperbolic Models

3.1.1 The Poincaré Disk Model

Henri Poincaré, a French mathematician, philosopher and physicist created this model which is considered as the easiest model to understand of all the models of hyperbolic geometry. This hyperbolic geometry is also called the *Poincaré ball model* (in three or more dimensions) or the *Conformal disk model* (in two dimensions). Hyperbolic geometry can be conformally represented in Euclidean n-space. One such way is to represent the hyperbolic plane as the points inside a ball. This is an n-dimensional hyperbolic geometry in which points of the geometry are in n-dimensional Euclidean geometry.

A *Point* in this model has the same meaning as in Euclidean geometry whereas a *line* in hyperbolic plane can be defined in two ways. One type of line is represented by open diameters meaning open chords passing through center of the ball, and the second type of line is any open arc of a circle that is orthogonal to the sphere which is the boundary of the ball. And the terms “lie on” and “between” have the same meaning as in Euclidean geometry. In the following figure, D1 is the first type of line that passes through the center of the circle and D2 and D3 are the second type of lines which are open arcs [7].

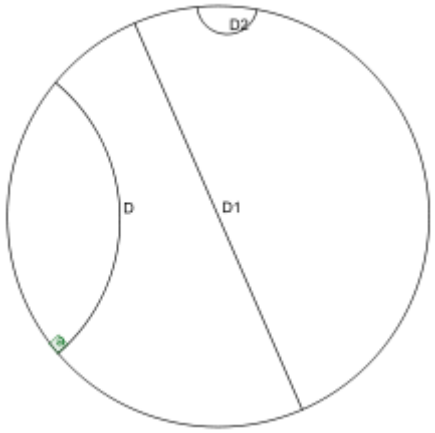

Figure 3.1 Lines in Poincaré Model

3.1.2 The Beltrami-Klein Model

The German mathematician Felix Klein was the first to propose this model. This is also called as Klein model. This is similar to Poincaré model. But this model is not a conformal model which means that the angles are not accurately represented by the model. This is also an n-dimensional hyperbolic geometry in which points of the geometry are in an n-dimensional Euclidean ball.

As in Poincaré, the *point* on this plane has the same meaning as in Euclidean geometry. And a *line* in the hyperbolic plane is defined as an open chord in this model. An open chord is any chord of the ball excluding the end points. And the terms “lie on” and “between” have the same meaning as in Euclidean geometry. In the following figure l, m and n are open chords [7].

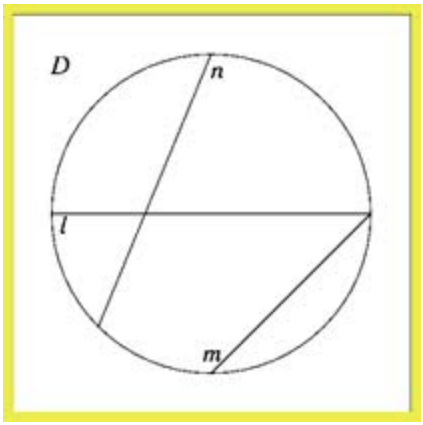

Figure 3.2 Lines in Klein Model

3.1.3 The Weierstrass Model

This model is the infinite model of the hyperbolic geometry. In this model the entire hyperbolic space is represented on the surface of the hyperboloid. A hyperboloid is a cone-like structure which is 3 dimensional. The advantage of this model is that the other two finite models can be obtained by the projections of this model.

The mathematical equation for the hyperboloid is $\mathbf{<X, X> = x^2+y^2-z^2 = -k^2}$ where **X** is a vector (x, y, z). This gives two parts or sheets for the hyperboloid: an upper sheet and a lower sheet. The points on lower sheet are reflections of the points on the upper sheet. Therefore the lower sheet is discarded. By doing so, a new mathematical equation $\mathbf{<X, X> = -K^2}$ and z> 0 is obtained to represent this single sheet [9]. The asymptotic cone bounding the upper sheet consists of all the points that satisfy **<X, X> = 0**.

In this model, *point* is defined as the point X that satisfies the equation of the upper sheet which is $\mathbf{<X, X> = -K^2}$ and z> 0. Whereas the *line* is defined as the section where a plane passing through the origin intersects the hyperboloid. If the point satisfies the equation **<X, L> = 0** and z>0 then L “lies on” the plane. A line R is said to be in “between” P and Q if R can be expressed as a linear combination of P and Q. The brown line in the following figure represents the line of this model.

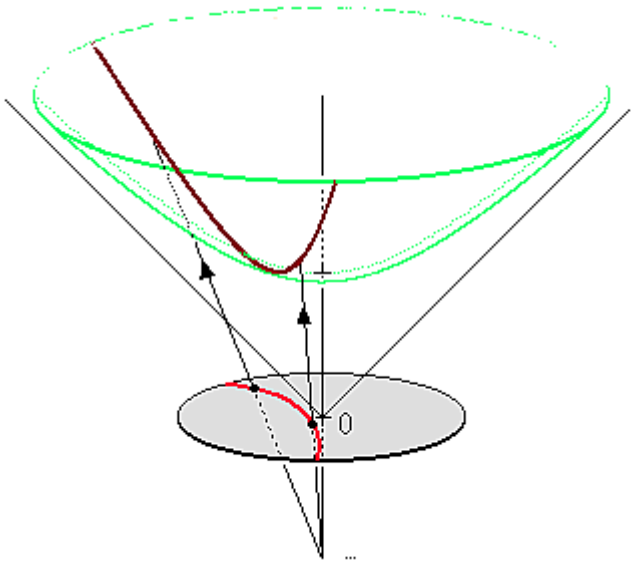

Figure 3.3 Lines in Weierstrass Model

3.2 Isomorphism

An isomorphism is a mapping from one model to another model that preserves corresponding structures. The lines and points in one model correspond to the lines and points in other model, and the relation between these models is obtained using this isomorphism. The Weierstrass model can be projected to obtain the Klein and the Poincaré models, thus this model is isomorphic to the other two models [7].

3.2.1 Weierstrass - Poincaré model Isomorphism

All the transformations in the program are currently done using this Weierstrass and the results are displayed using the Poincaré model. So, it is important to focus

on the concepts of isomorphism between these two models. In the Figure 3.3 the red line on the circle is the corresponding Poincaré line for the brown line which is Weierstrass line.

The Poincaré model is obtained by stereographic projection of the 3D Weierstrass model onto the x-y plane. The projection is towards the point (0, 0, -1). It is given by

[x, y, z]1/(z+1) [x, y, 0]

The Poincare to Weierstrass inverse projection is given by

[x, y, 0] 1/(1 - x^2– y^2) [2x, 2y, 1 + x^2+ y^2]

3.2.2 Weierstrass – Klein model Isomorphism

The Klein model is obtained by stereographic projection of the Weierstrass model onto the z=1 plane. The projection is directed towards the point (0, 0, 0). It is given by

[x, y, z] [x/z, y/z, 1]

The Klein to Weierstrass inverse projection is given by

[x, y, 1] 1/ 1-x2-y^2[x, y, 1]

Chapter 4

Tessellations and Hyperbolic Patterns

The algorithms and the methodologies used to create the patterns of interest are discussed in this chapter. The tessellations and patterns are explained in the first two sections. The symmetry operations used to generate these patterns are discussed in the next session. The following sections defines the motif and a fundamental region, the pattern generation algorithm, and the last session describes the implementation of the replication algorithm which is used in the pattern generation algorithm.

4.1 Tessellations

A tessellation is defined as the pattern formed by arranging the translated and transformed congruent copies of the basic sub-pattern in a mosaic fashion. The word “tessellate” is derived from the ionic version of the Greek word “tesseres”, for which English meaning is “four”. In Figure 4.1 some examples of these tessellations are shown.

4.2 Repeating Hyperbolic Patterns

A *repeating pattern* is obtained by transforming and translating the basic sub-pattern called the *motif*. The scope of the current thesis is repeating hyperbolic patterns, which are made of hyperbolically congruent copies of motif. We will be

denoting “repeating hyperbolic patterns” as “repeating patterns” from this point. An example for such

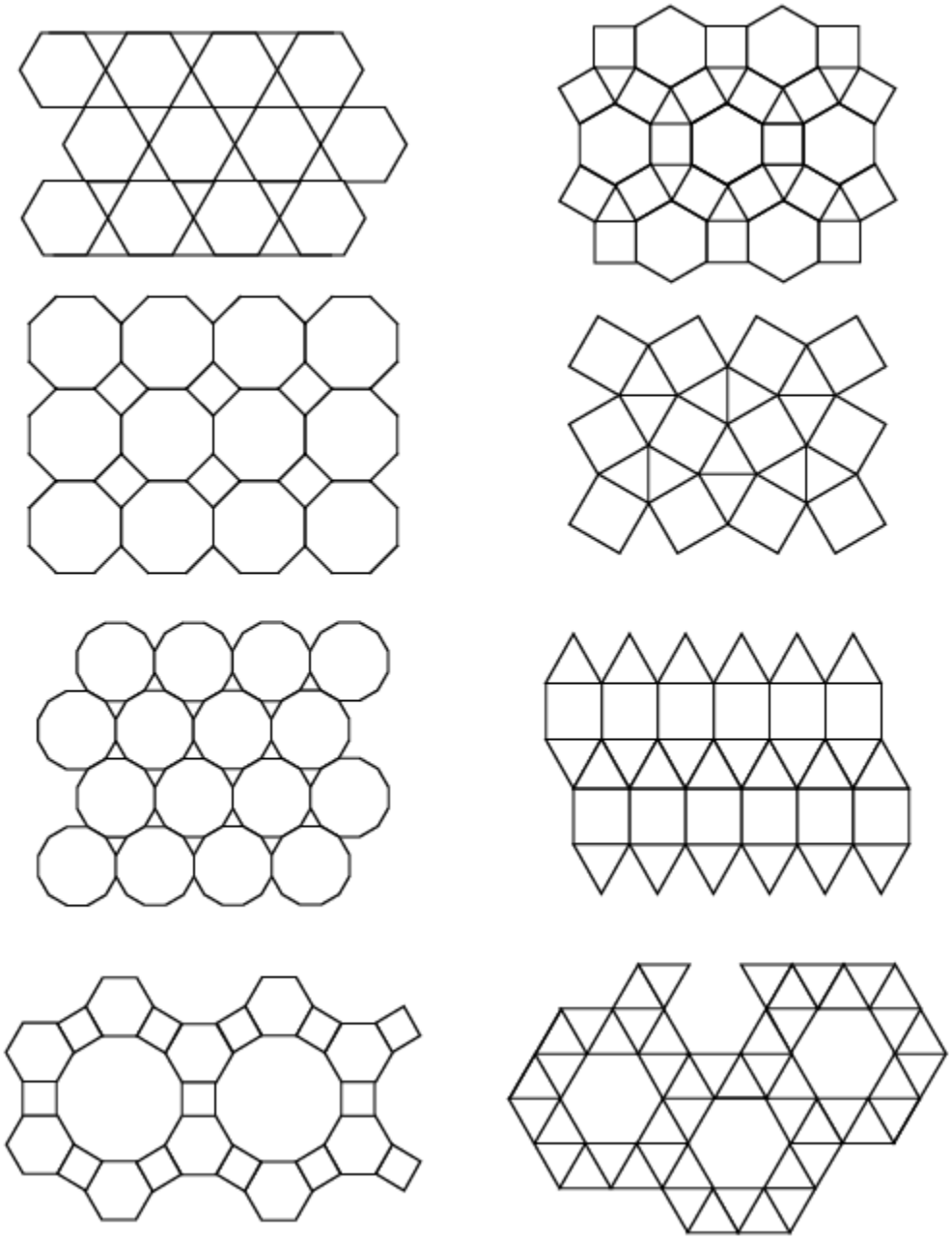

Figure 4.1 Euclidean tessellation examples

repeating hyperbolic pattern is shown in Figure 4.2. This repeating hyperbolic pattern is a regular tessellation {p, q} of the hyperbolic plane. The notation {p, q}

says that it has regular p-sided polygons and q congruent copies of these polygons meet at each vertex. The polygons can be irregular also. But the focus of this thesis is limited to regular polygons only. The example shown in the Figure 4.2 has a {6, 4} tessellation. So, {p, q} is used to denote a regular tessellation from this point.

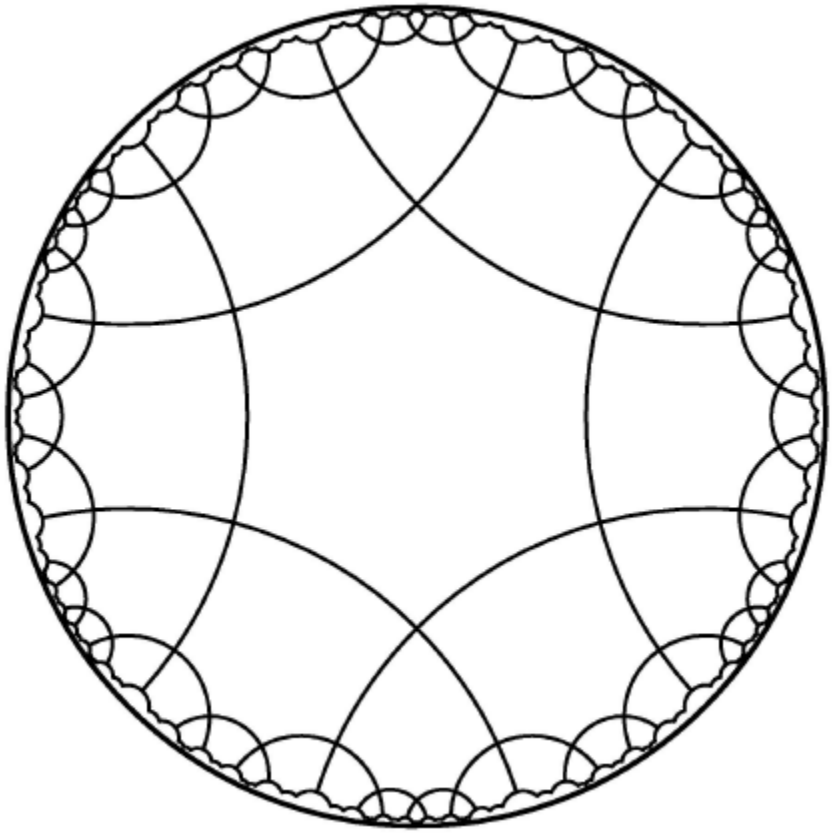

Figure 4.2 The regular tessellation {6, 4} of the hyperbolic plane

Also for a tessellation {p, q} to be in hyperbolic plane, it has to satisfy the condition (p-2) (q-2) > 4. The tessellations with (p-2) (q-2) < 4 are Spherical, and tessellations with (p-2) (q-2) = 4 are Euclidean tessellations. An example of the computer generated repeating hyperbolic pattern is shown in Figure 4.3.

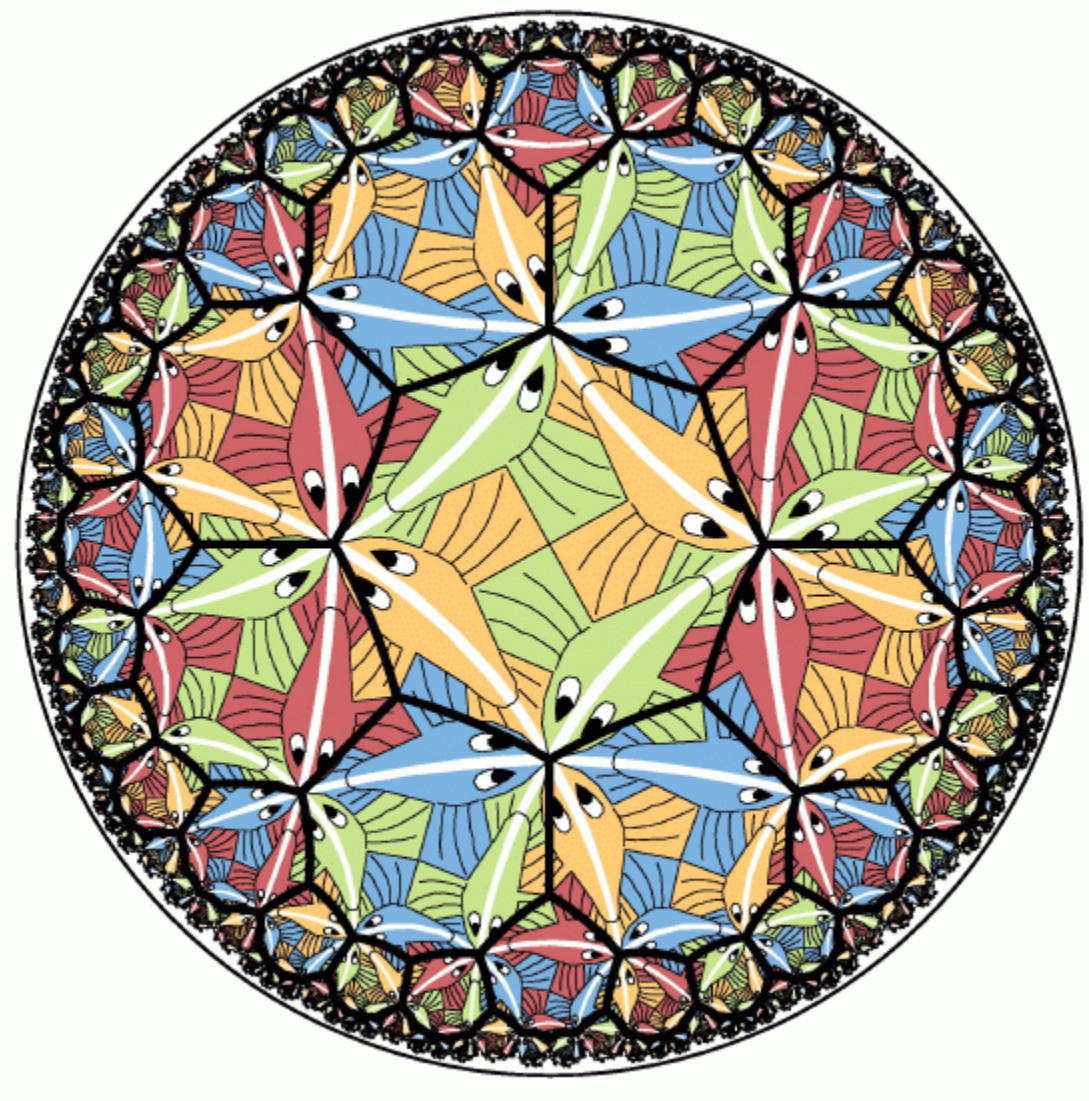

Figure 4.3 A computer generated version of Escher's *Circle Limit III* showing the {8, 3} tessellation

4.2.1 Symmetry Groups

A *Symmetry operation* is defined as an isometry that transforms a pattern onto itself. It is a distance preserving mechanism between two metric spaces. A symmetry operation is also called "symmetry" in short. And the set of all these symmetry operations is called the *Symmetry group.*

The repeating patterns are divided using fixed lines of reflection called mirrors. These mirrors of a {p, q} tessellation divide the each p-gon into 2p right angled triangles in which each triangle has acute angles of π/p and π/q. As this is hyperbolic geometry, the sum of all the angles is less than 2π. The Symmetry group of the regular tessellation {p, q} is denoted by [p, q]. This can be generated by the reflections across the 3 sides of the right angled triangle. An example of the symmetry group [6, 4] for the tessellation {6, 4} is shown in Figure 4.4.

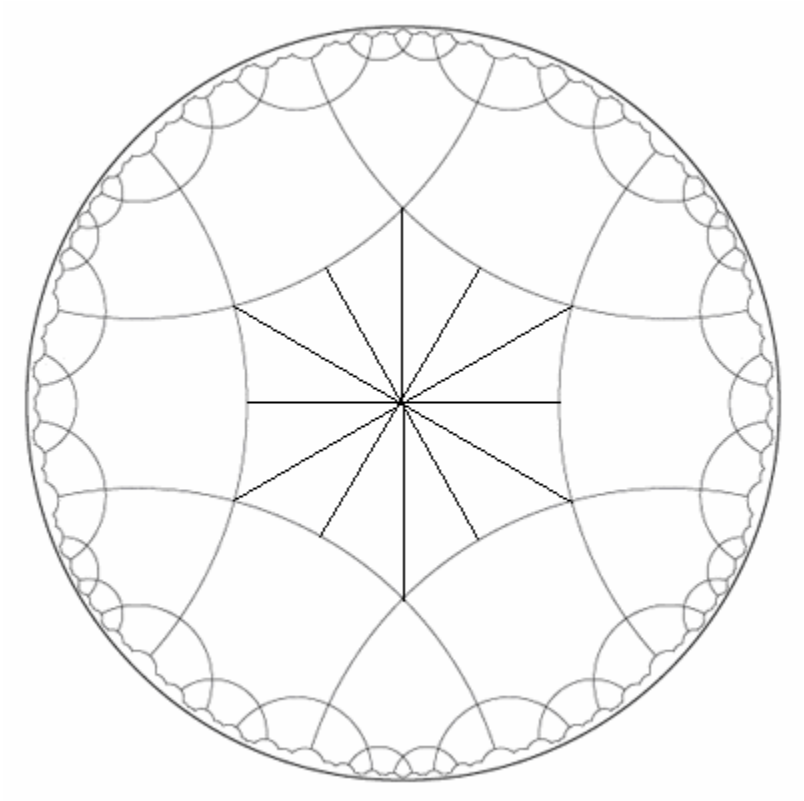

Figure 4.4 A pattern with symmetry group [6, 4]

Another symmetry group $[p, q]^{\cdot}$, is a subgroup of [p, q] of index 2[10] and there are at least two ways to generate the repeating patterns using this symmetry. First method to generate is to take the even number of reflections which were a part of

the [p, q] symmetry group. And the second method to generate these patterns is by any two rotations of 2/p, 2/q, about the corresponding vertices of the right angled triangle formed by the lines of symmetry. An example of a pattern formed using this symmetry group [p, q]· is shown in Figure 4.5

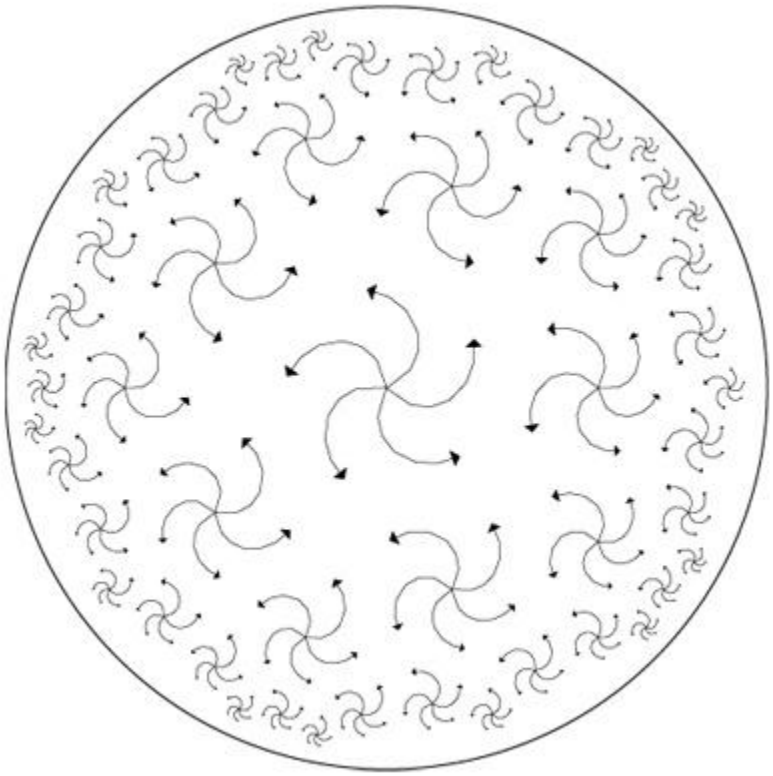

Figure 4.5 An example pattern of symmetry group [p, q]·

Another type of symmetry group [p·, q] is also a subgroup of [p, q] of index 2[10]. A repeating patterns can be generated using this symmetry group by rotations of 2/p about the center of p-gon and reflections across the edges of the regular tessellation {p, q}. An example for this kind of symmetry group is shown in the following Figure 4.6.

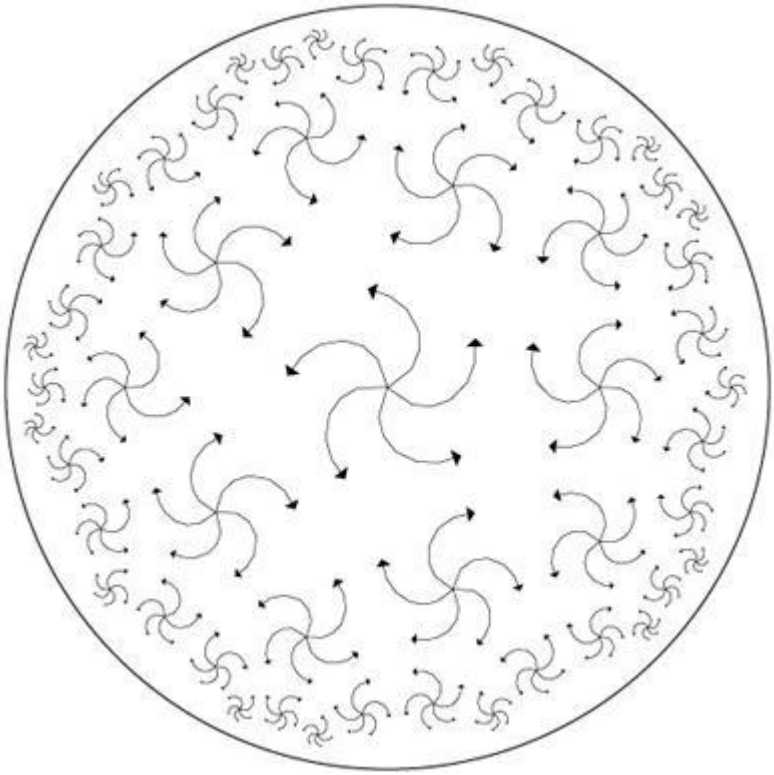

Figure 4.6 An example pattern of symmetry group [p, q]

4.2.2 Motif and Fundamental Region

A *motif* is the basic sub-pattern that is used to generate the repeating pattern. As defined by Dunham [10], if the hyperbolic plane is covered without overlapping the transformed copies of a connected set under elements of a symmetry group, that set is called *fundamental region* for the symmetry group. If motif covers the entire fundamental region, then the repeating hyperbolic pattern generated will be interlocking. An example for this kind of pattern is shown below.

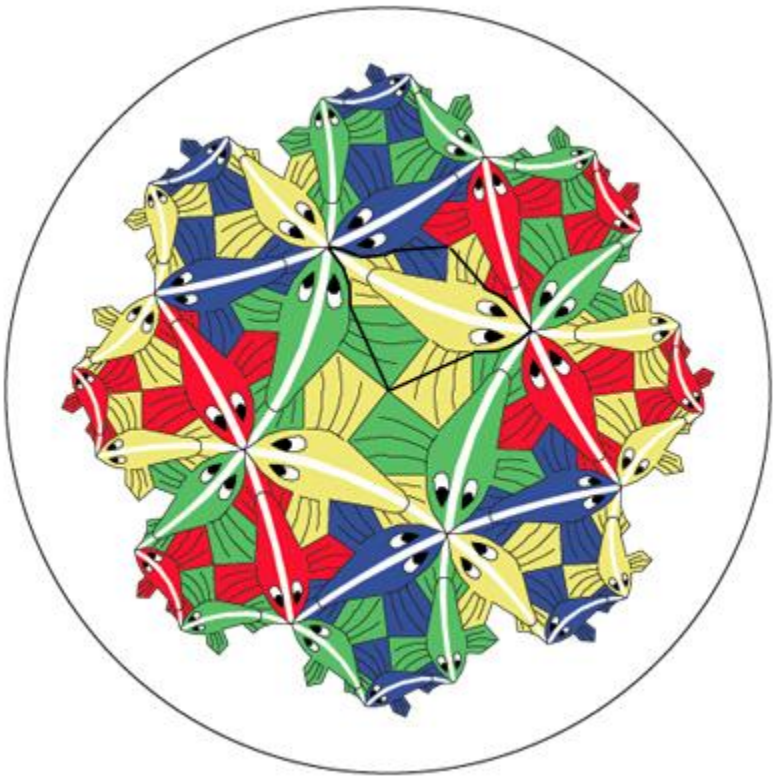

Figure 4.7 Fundamental region shown in dark boundaries, an interlocking repeating pattern

4.2.3 Repeating Pattern Generation Algorithm

Dunham designed an algorithm to create these repeating patterns [4] as a part of his research work. These patterns are designed by replicating the basic sub-pattern motif. This involves two steps. The first step being creating the *central p-gon pattern* by replicating the fundamental region. This pattern becomes the first layer of the repeating pattern. By rotating the motif around the p-gon center and/or reflecting it across the diameters and perpendicular bisectors of the edges until the p-gon is filled with copies of motif. The following figure show the creation of the first layer of the repeating pattern.

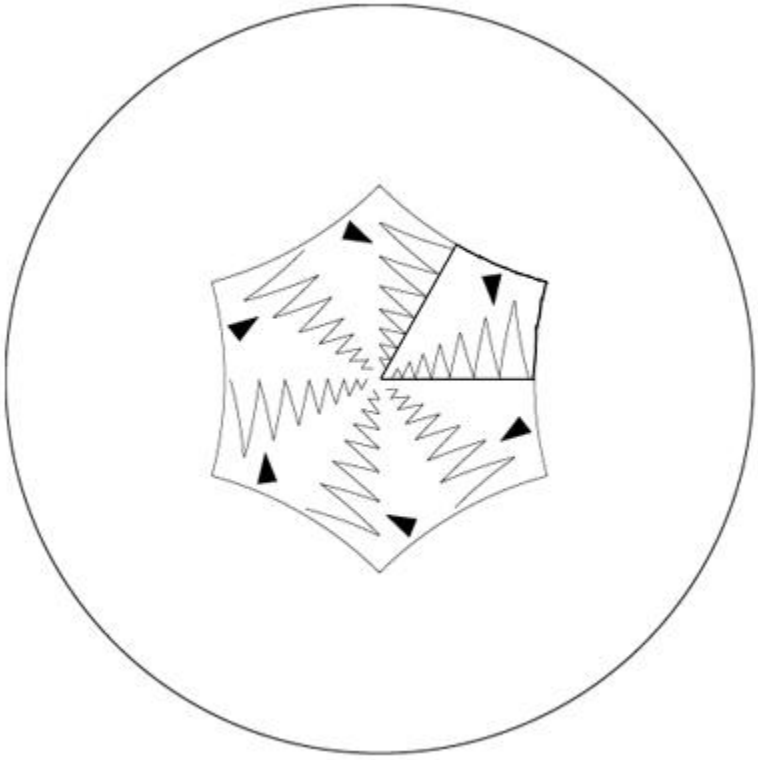

Figure 4.8 The fundamental region replication to form the central p-gon pattern

The second step is to replicate the p-gon pattern in order to obtain the complete repeating pattern. This replication of p-gon pattern is advantageous over the replication of motif which makes the algorithm simpler by reducing the number of transformations, efficient and less susceptible to the hardware errors. The layers are generated recursively. The first step of this algorithm generates the first layer of the pattern and *k+1* layer consists of all the p-gons sharing an edge or vertex with *k* layer.

Figure 4.9 shows the extending of p-gon pattern from layer 2 to layer 3. The p-gon 1 is rotated about vertex A to draw p-gon 2. It is rotated about B to get p-gons 3 and it is again rotated about C to get p-gons 4.

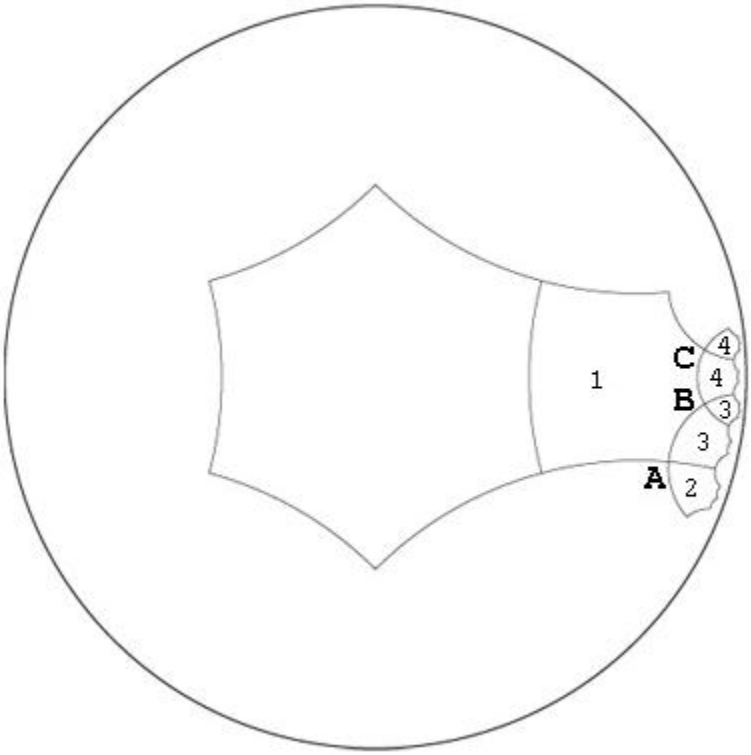

Figure 4.9 Extending pattern from layer 2 to layer 3

The transformation of a p-gon from one layer to next is done by applying the symmetry operations on all the points of the fundamental region. Each point is projected onto the hyperboloid of the Weierstrass model by the inverse projection. 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. Then it is projected back to the Poincaré model.

4.2.4 Implementation of Replication Algorithm

The second step of creating the repeating patterns, which is recursive replication algorithm, is described in this section. To obtain the *(k+1)*th layer from *k*th layer, the algorithm iterates over each vertex of the p-gon in the kth layer which it shares with (k+1)st layer. For each vertex, it calculates the number of polygons needed to draw the next layer from that vertex. It can be either *q - 2* or *q - 3* depending upon the exposure of the vertex. Then the algorithm is recursively called for the vertices of the p-gons in the newly formed layer. As shown in Figure 4.9, after p-gon 1 is drawn, the replication algorithm is called for each of its vertices A, B and C. The algorithm is then recursively called for all the exposed vertices of the new p-gons. The software architecture of the program is explained in Appendix B.

Chapter 5

Graphical User Interface

This chapter demonstrates the graphical user interface of the program which performs various functionalities such as replicating the patterns using the existing data files, creating new patterns, modifying existing patterns, and saving newly created patterns in a consistent data file format.

Figure 5.1 shows how the user interface of the applet looks like when it is started. It contains a drawing canvas, on which all the repeating hyperbolic patterns are generated. It contains a menu bar which is used to open the existing data files, modify the existing patterns, create new patterns, and save the patterns created. Figure 5.2 shows the File menu under which the menu items: new, open, save and exit operations are included.

The “New” menu is used to open a dialog box in which to enter the number of sides of the central polygon (p), the number of polygons meeting at each vertex (q), the number of different sides in the central p-gon that are used to create a fundamental region, the maximum number of colors, the kind of reflection symmetry, and the transformation. Figure 5.3 shows what this interface looks like. A sample of input given to this interface is shown in Figure 5.4.

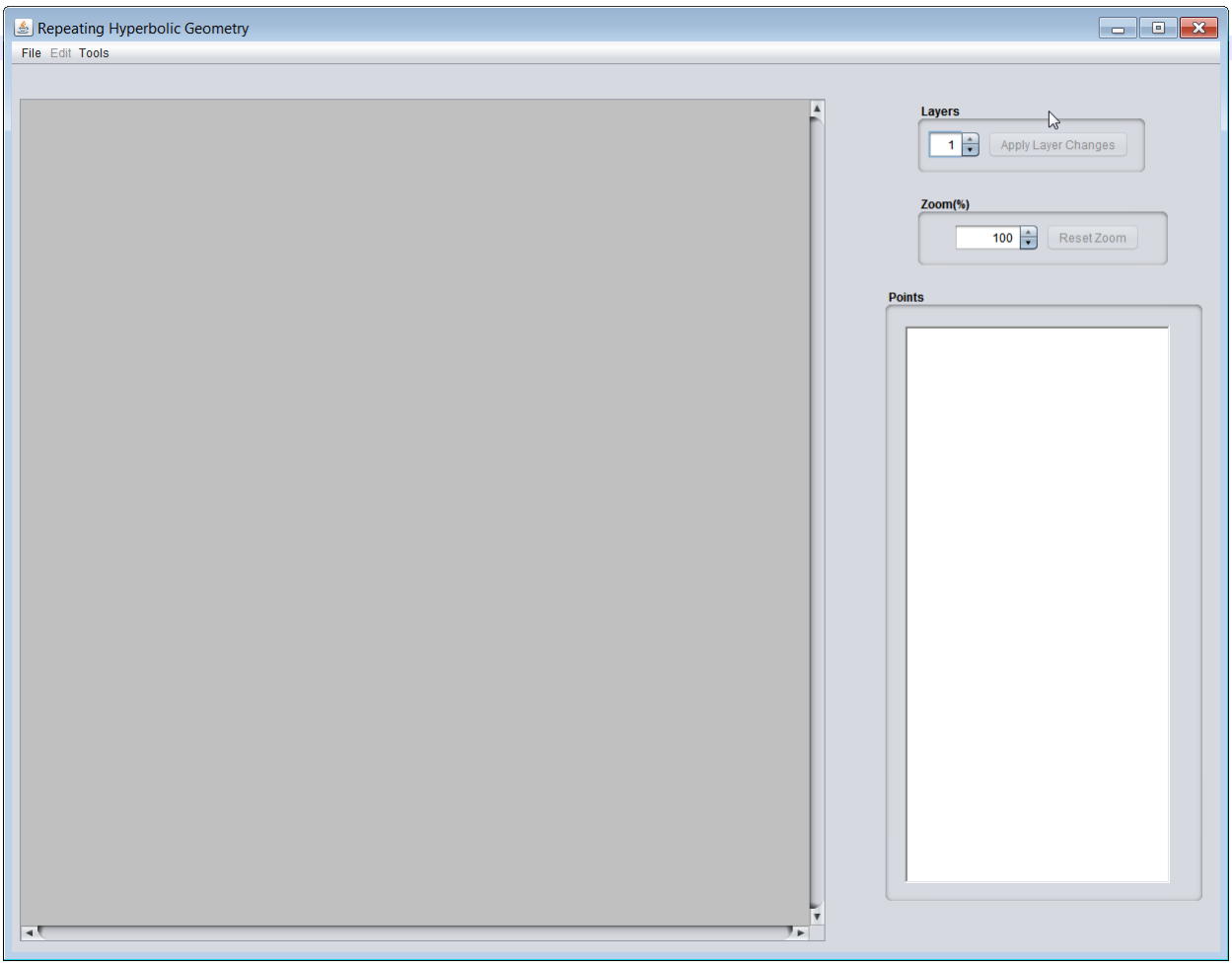

Figure 5.1 Screenshot of the Applet when it starts running.

After executing the information given to the “New” menu item interface, the central polygon is drawn on the applet’s drawing canvas. Figure 5.5 demonstrates the sample polygon that is created, based on the input given in Figure 5.4. This polygon is based on {6, 4} pattern. The fundamental region or the motif created will be part of the central polygon created.

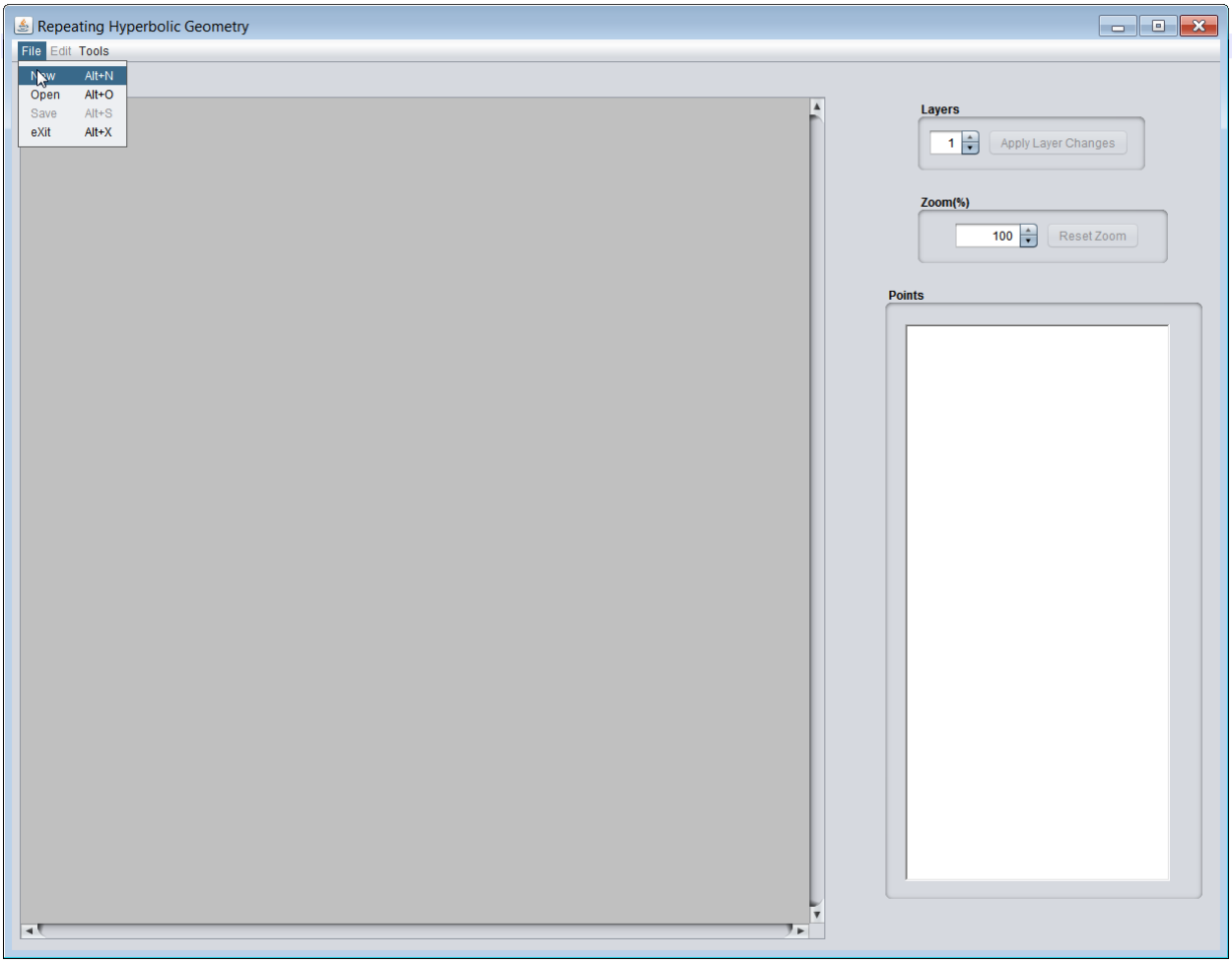

Figure 5.2 Screenshot showing the menu bar for opening a Dialog to create a new pattern.

The “Edit” menu is used to add various points to draw variety of patterns such as circles, filled circles, filled polygons, filled p-gons, and polylines. Depending on the shape used, the required number of points can be added to the pattern using the “add point” button. Each point has an X-coordinate value and Y-coordinate value scaled according to the size of the drawing canvas. This functionality is

demonstrated in Figures 5.6 and 5.7. The required color combinations for the patterns can be chosen using this dialog box.

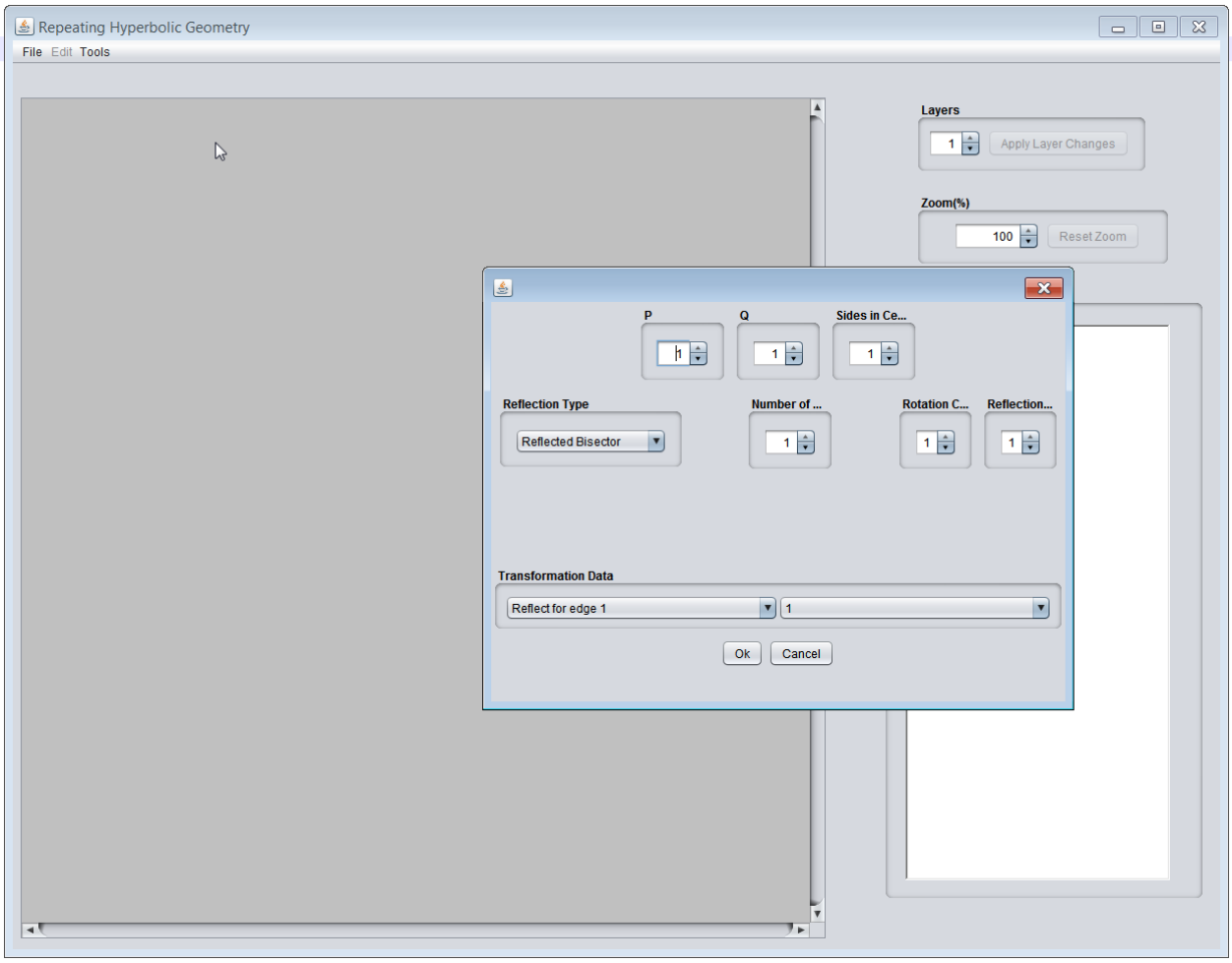

Figure 5.3 Screenshot of Dialog box that accepts required parameters for creating pattern.

A sample of a fully formed {6, 4} pattern is shown in Figure 5.8. The patterns formed are shown on the drawing canvas of the user interface. Similarly, by using the “Open” menu item of the File menu, already existing data files can be opened

and read to generate repeating hyperbolic patterns. The central motif is created first and then it is replicated according to reflection symmetry properties specified to form the central p-gon pattern. Instead of replicating the fundamental region or the motif, this central polygon is replicated which increases the efficiency of this pattern generation.

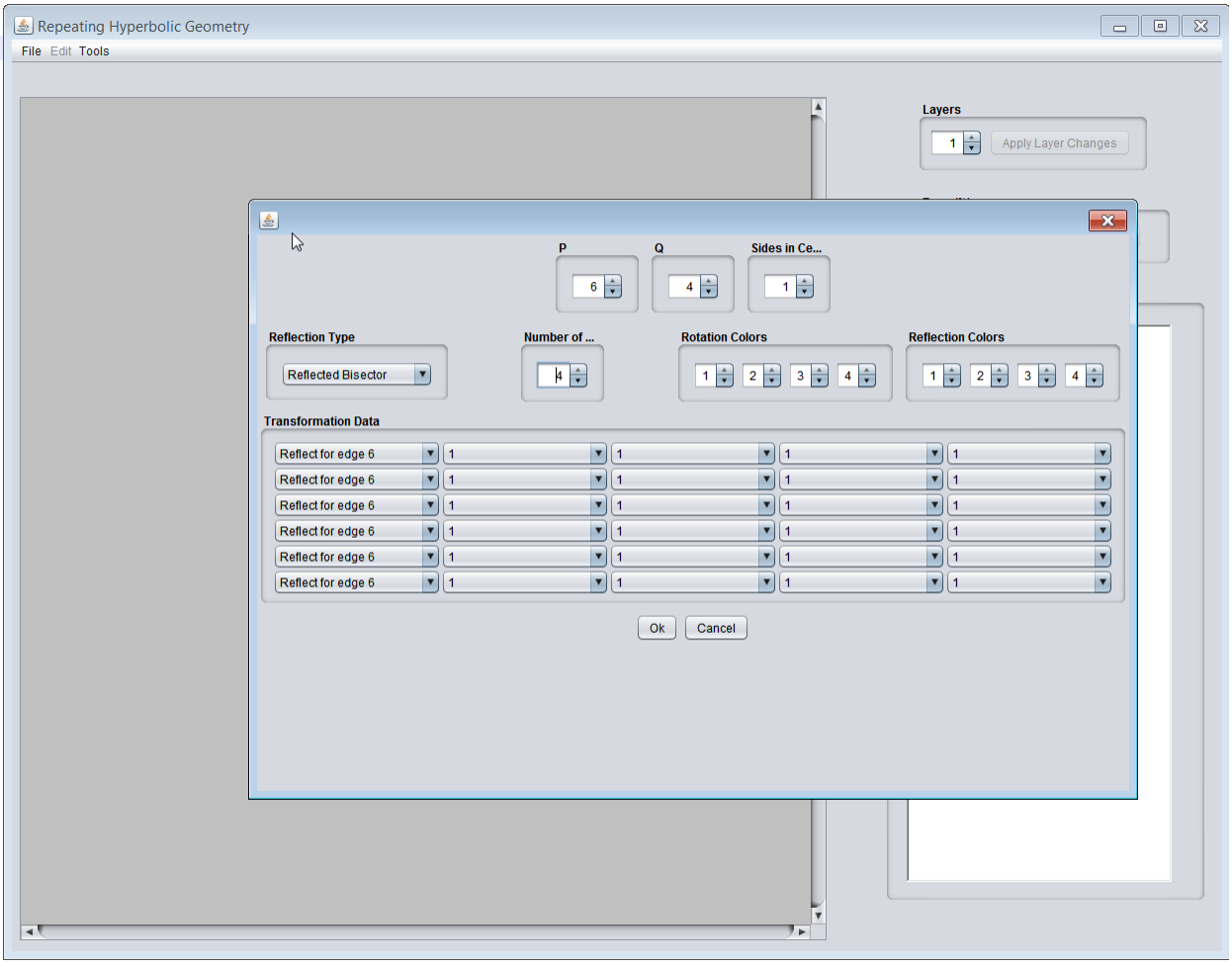

Figure 5.4 Sample input to draw a {6, 4} pattern.

The sample central p-gon generated is shown in Figure 5.9. This is a single layer of the hyperbolic pattern and this can be replicated to as many layers as given in

the “layers” spinner that is shown on the user interface of the applet. The central p-gon pattern is repeated over 4 layers and is shown in Figure 5.10. This is another functionality of the applet to repaint the canvas with the pattern repeated to the number of layers as specified in the “layers” spinner which is located on the top right corner of the applet UI.

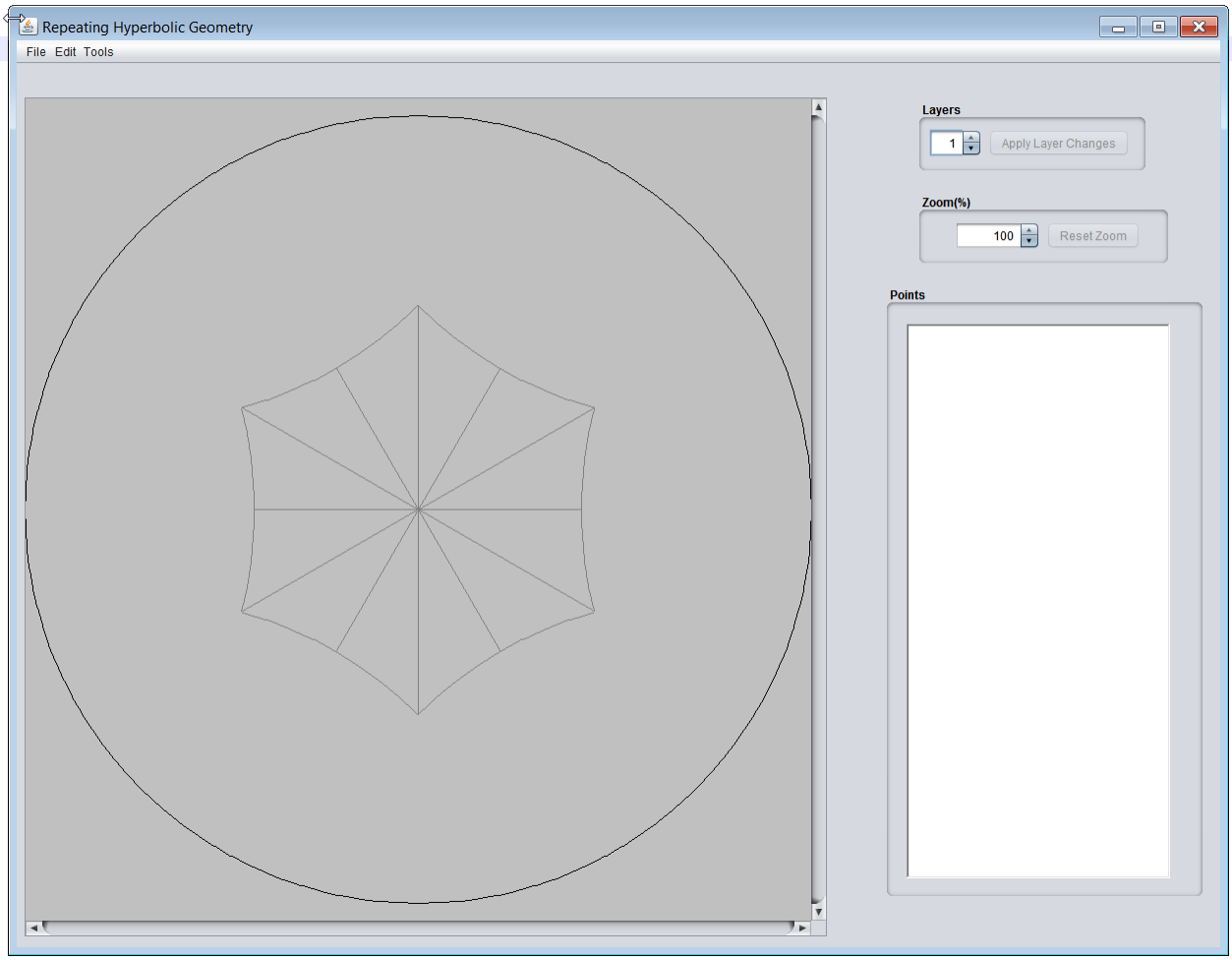

Figure 5.5 Outline of a {6, 4} pattern.

Figure 5.10 shows a 4 layer expanded pattern based on the {8, 3} tessellation. Also the user can also zoom in on the pattern using the “zoom box” located on the right side under the “layers” control box of the applet user interface. This can be

seen in Figure 5.11 in which the zoom level is specified as 125%. The user can zoom the image with a maximum zooming limit of 1000000% starting from 100%.

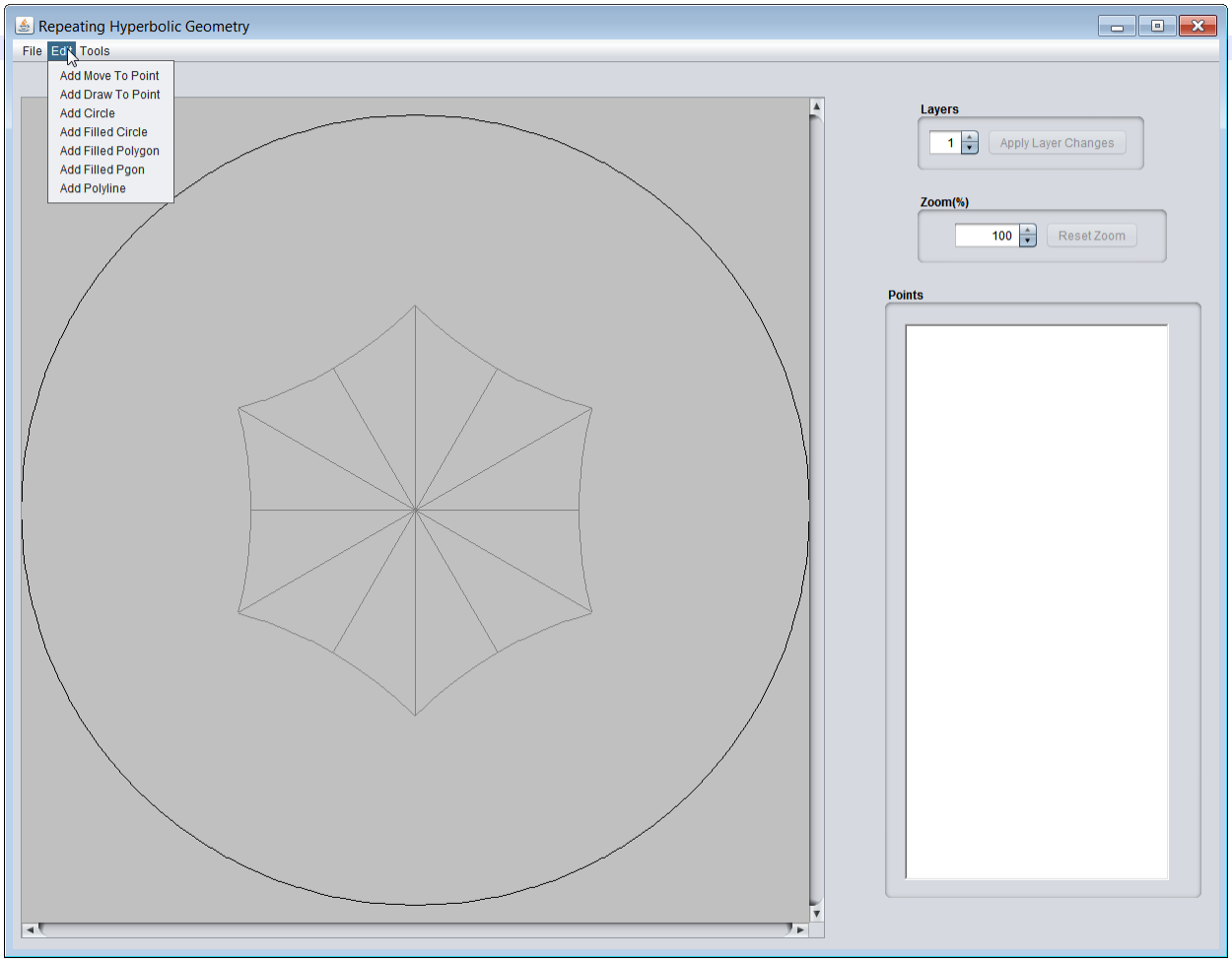

Figure 5.6 Screenshot showing the menu bar for opening a Dialog to add points to pattern.

The scroll pane which is located on the bottom right part of the applet lets the user look at the properties of the points that are used to draw the motif. The points are shown in the form of a tree, and on expanding those points, shows the properties of the points such as its x-coordinate, its y-coordinate, its color and the type of the

shape of which this point is part. Figure 5.12 shows a demonstration of this functionality.

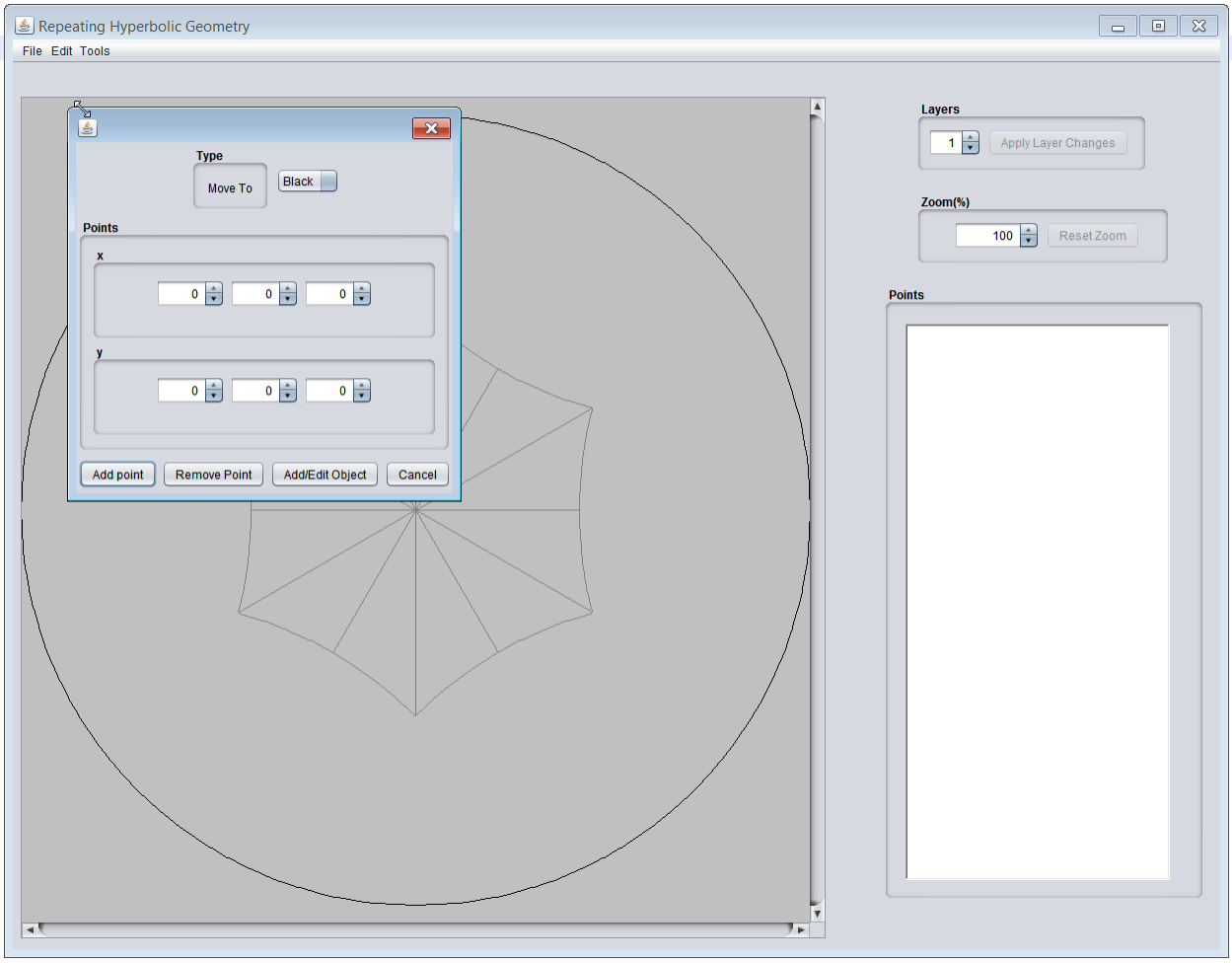

Figure 5.7 Screenshot of Dialog box that accepts points to create pattern.

The newly created points can be saved using the “Save” menu item under the “File” menu, which saves the data required to draw these patterns in a consistent format that is described in appendix. These data files can be read again to create the same patterns. The exit menu item which is located under file menu is used

to close this applet. These menu items can also be accessed using the shortcut keys that are shown in the screenshots of the user Interface of this applet.

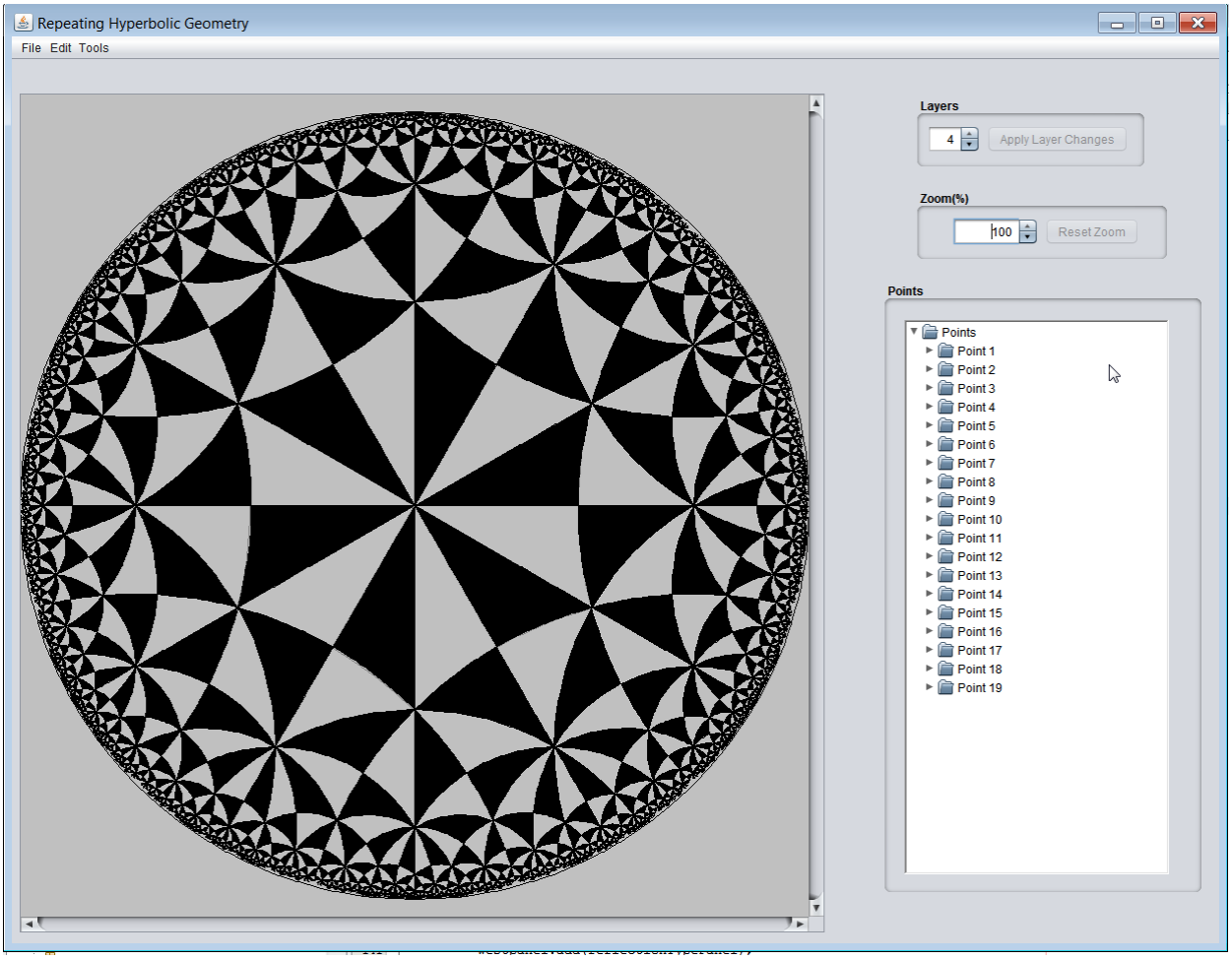

Figure 5.8 A repeating pattern based on {6, 4} with 4 layers.

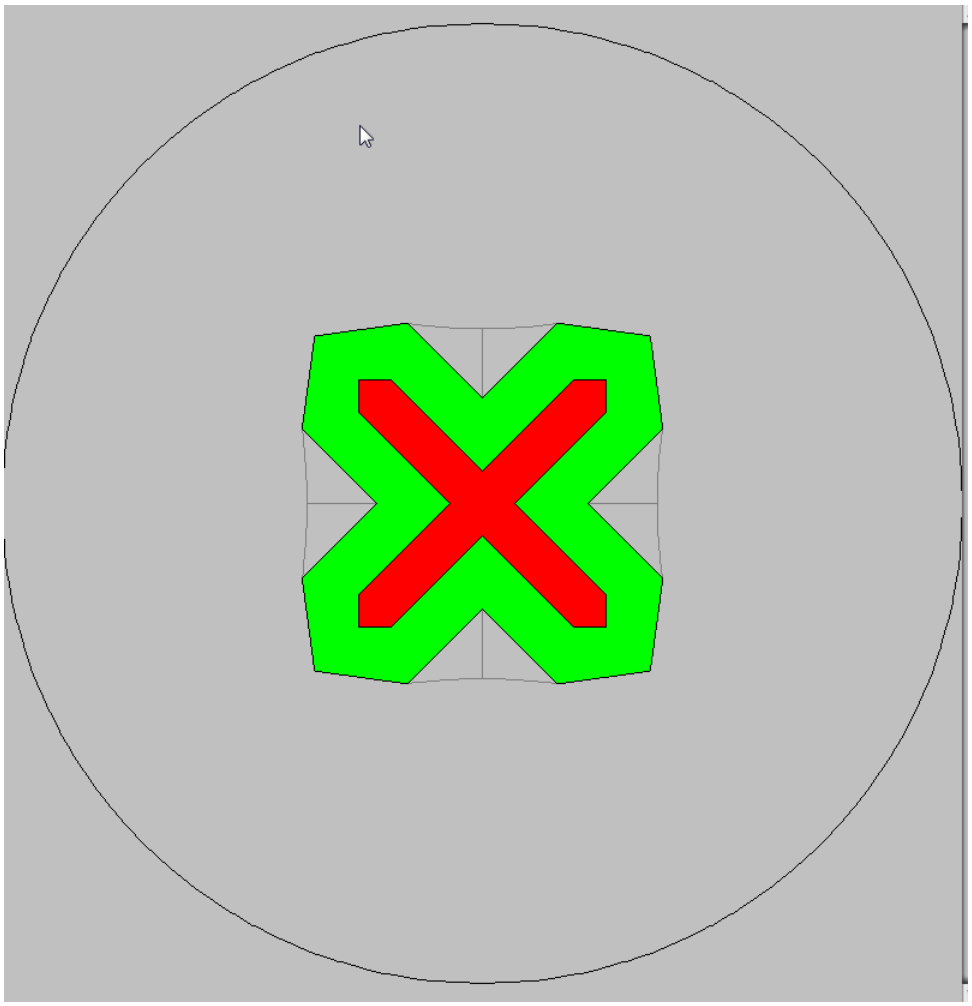

Figure 5.9 A {8, 3} pattern with single layer.

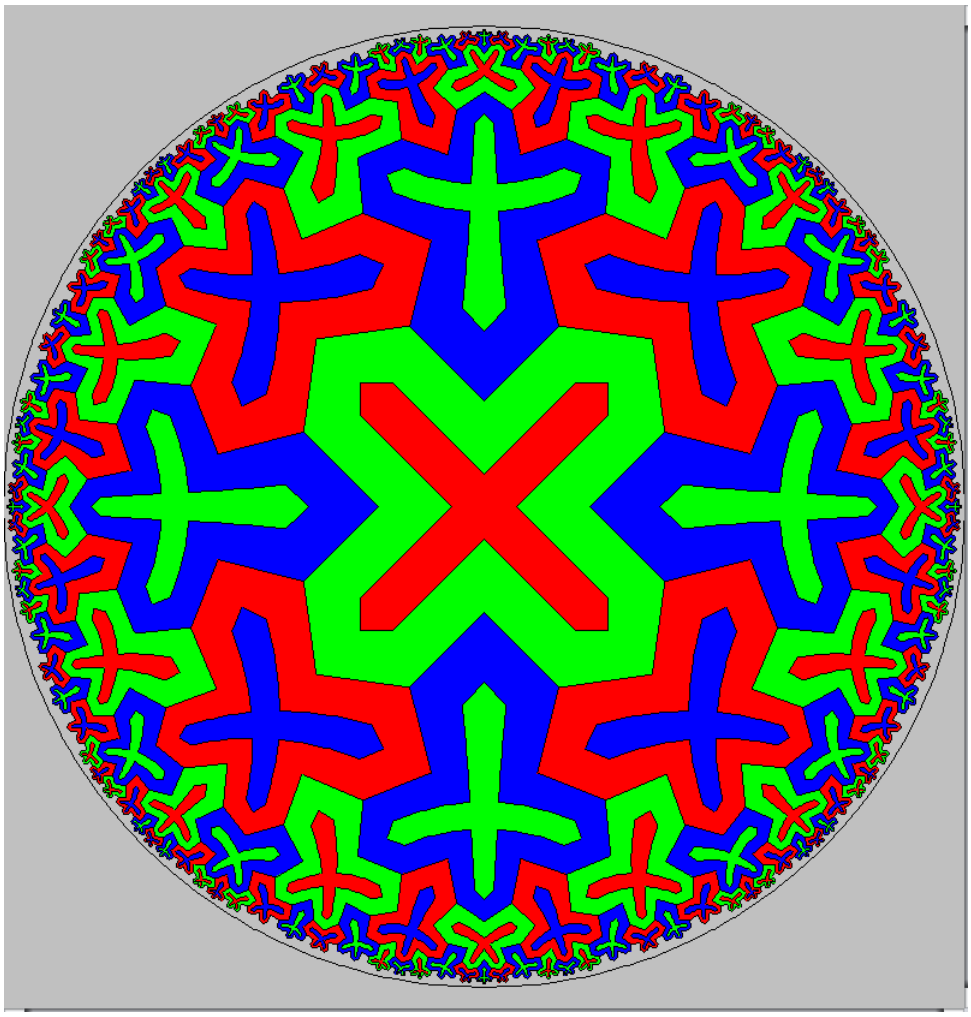

Figure 5.10 A {8, 3} pattern with 4 layers (showing multiple layer functionality).

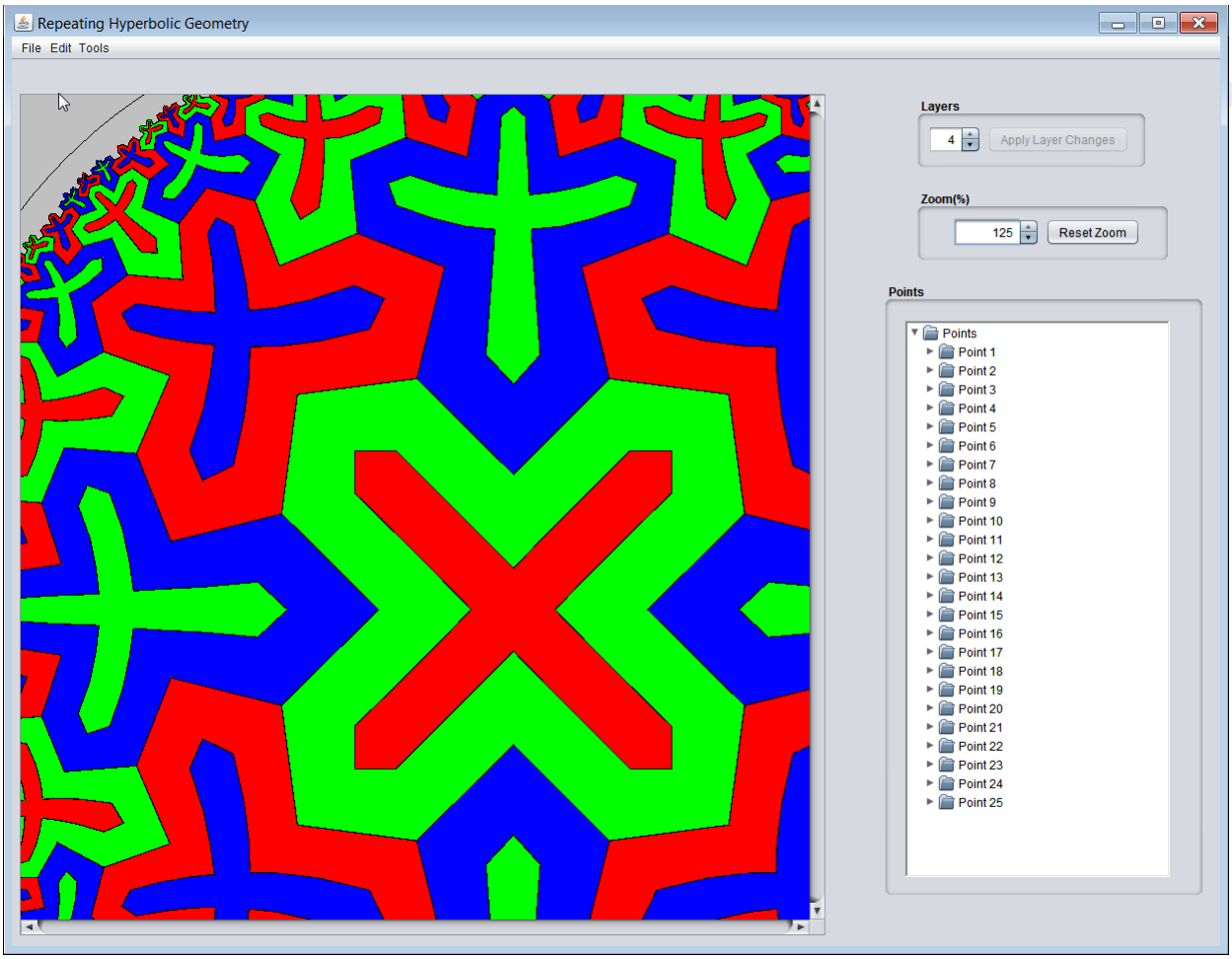

Figure 5.11 A {8, 3} pattern with 4 layers zoomed to 125% (demonstrating Zooming functionality).

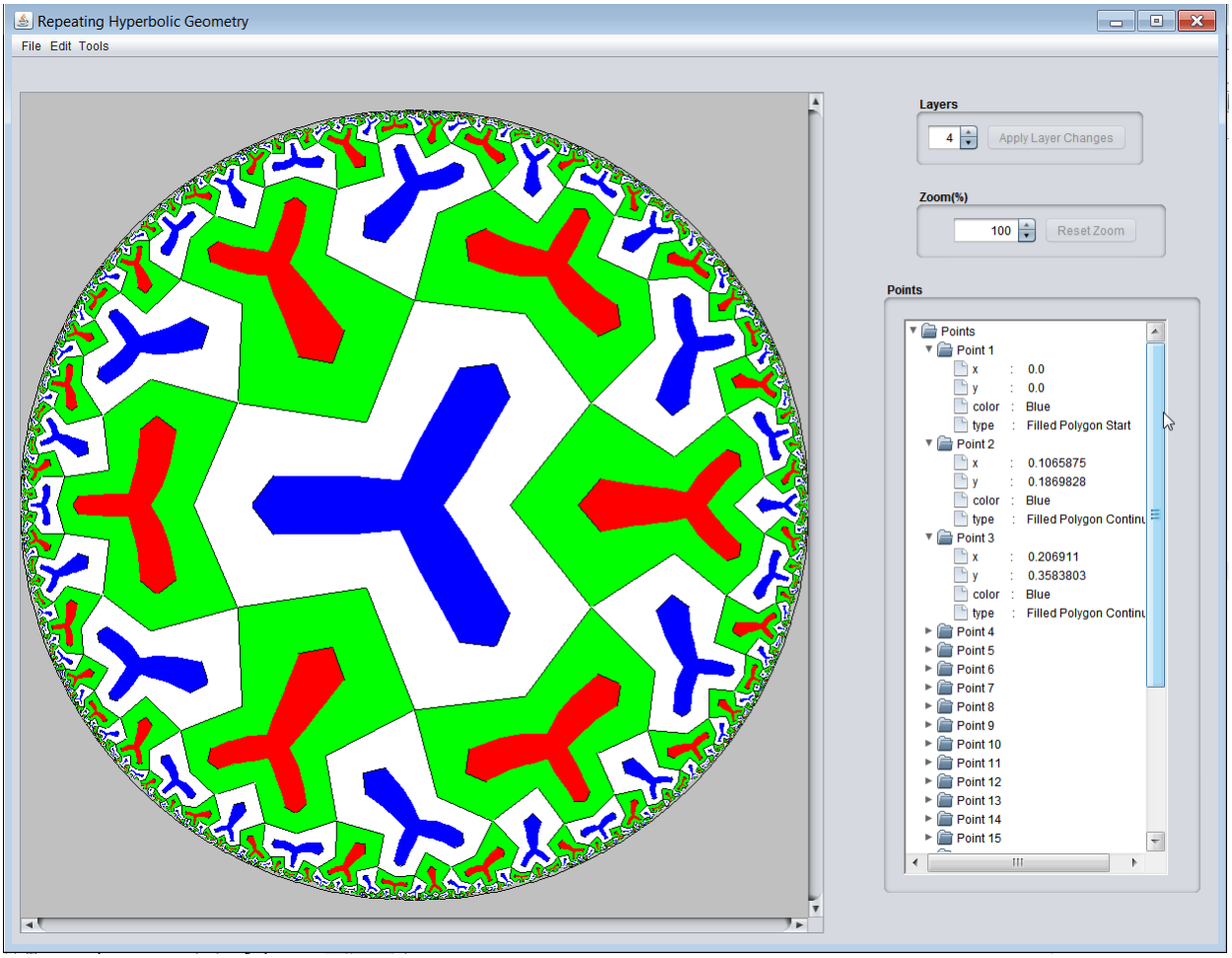

Figure 5.12 A {6, 4} pattern with points expanded (demonstrating point properties).

Thus this chapter gives an overview of what the user interface of this program looks like and also shows various functionalities of the program.

Chapter 6

Results

This chapter shows some of the results of the algorithm that is implemented using the Java applet. As this is a Java applet, it is portable to any platform and is easy to run, and the interface is user friendly. This program has improvements over the existing programs in terms of efficiency and cross platform compatibility. This “design” program was initially written in C using the Motif framework for the user interface, which requires it to run only on Linux or UNIX platforms. After that, another version of this program which was written in C++ using the Qt framework for the GUI, which is available only on certain platforms and is also complex to program. Several thesis papers were studied to understand the concepts of the algorithm. All of these programs are based on Dr. Dunham’s algorithm, but the current program has more error checking and was developed as an object oriented program, which is easier to read and understand.

All the existing data files have been validated using the current program; and the invalid files are identified which are not according to the format that is specified in the appendix. Some of the results for various data files are shown below. Some of the patterns portray Escher’s patterns. These are all the screenshots from the current applet.

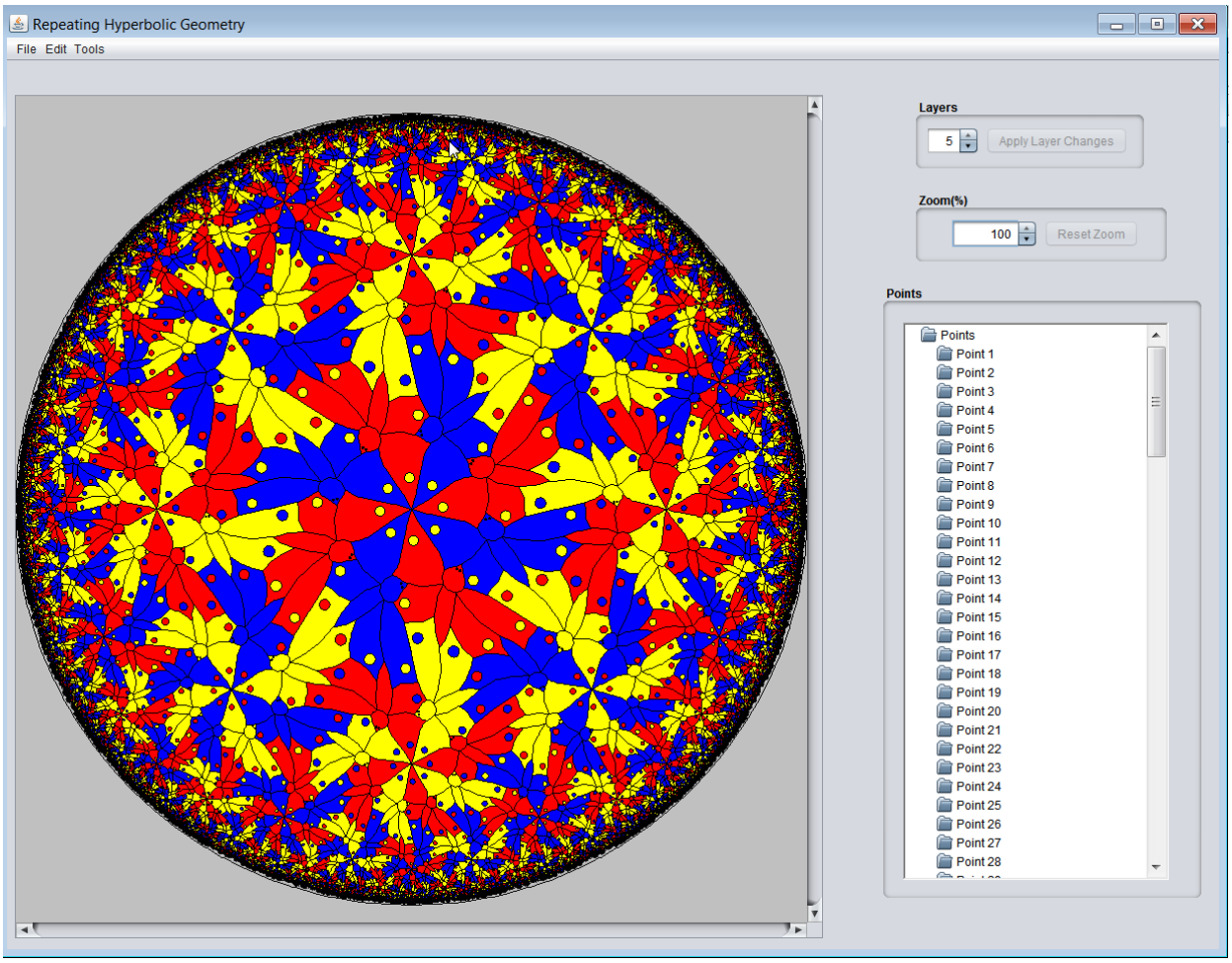

Figure 6.1 A Hyperbolic Butterfly Pattern Based on {8, 3} with 5 layers.

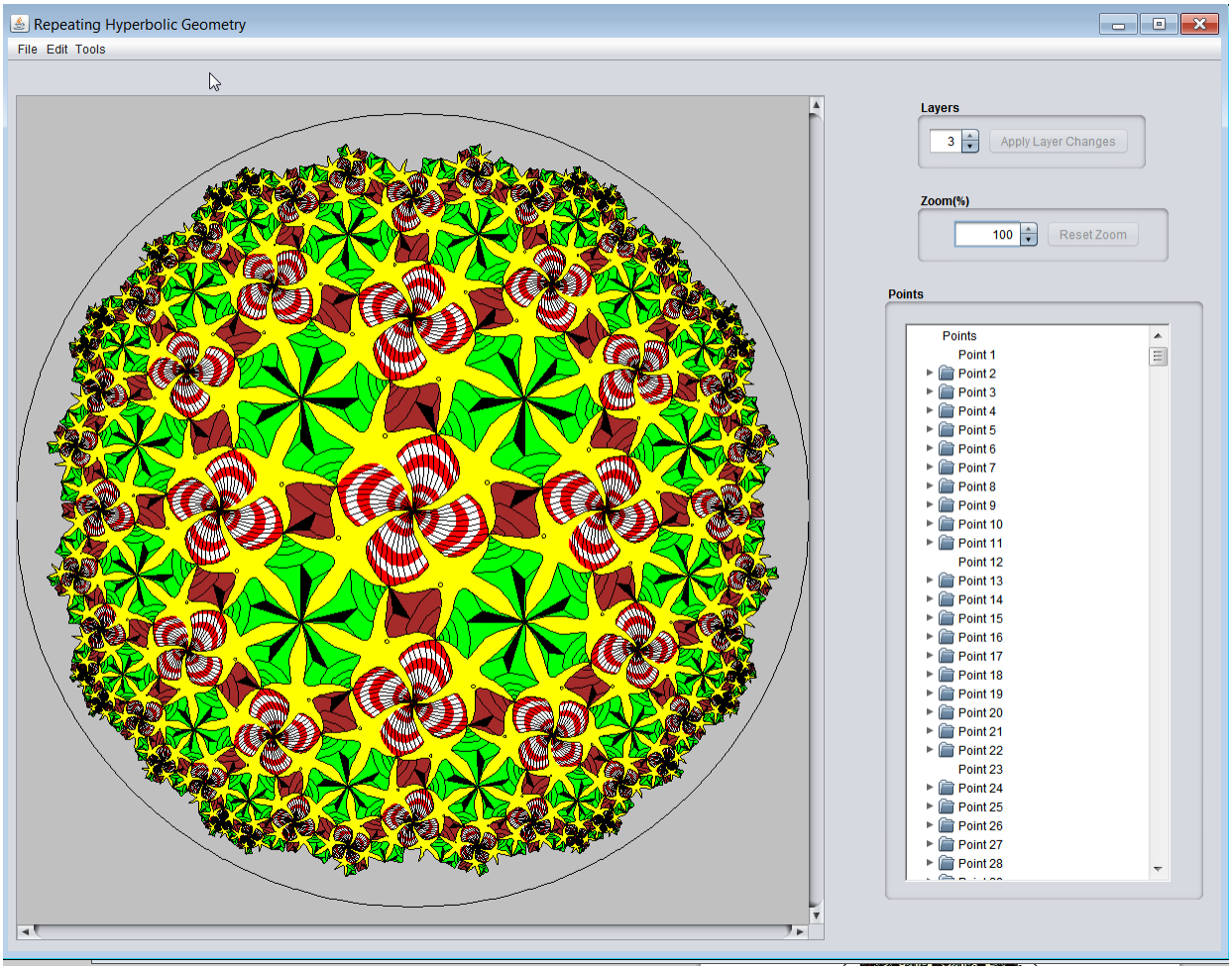

Figure 6.2 A Hyperbolic Shell Pattern Based on {4, 5} with 3 layers.

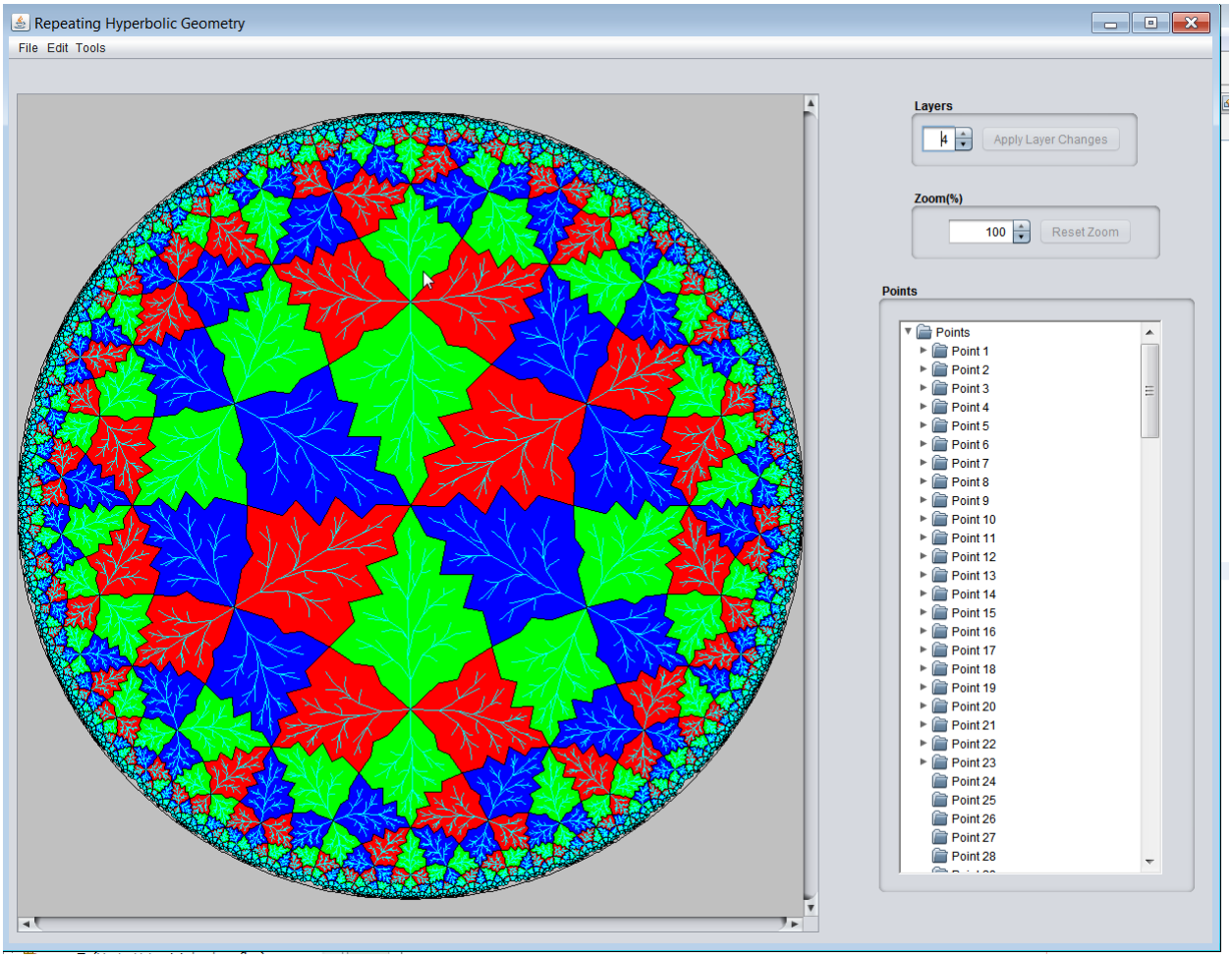

Figure 6.3 A Hyperbolic leaf Pattern Based on {6, 4} with 4 layers.

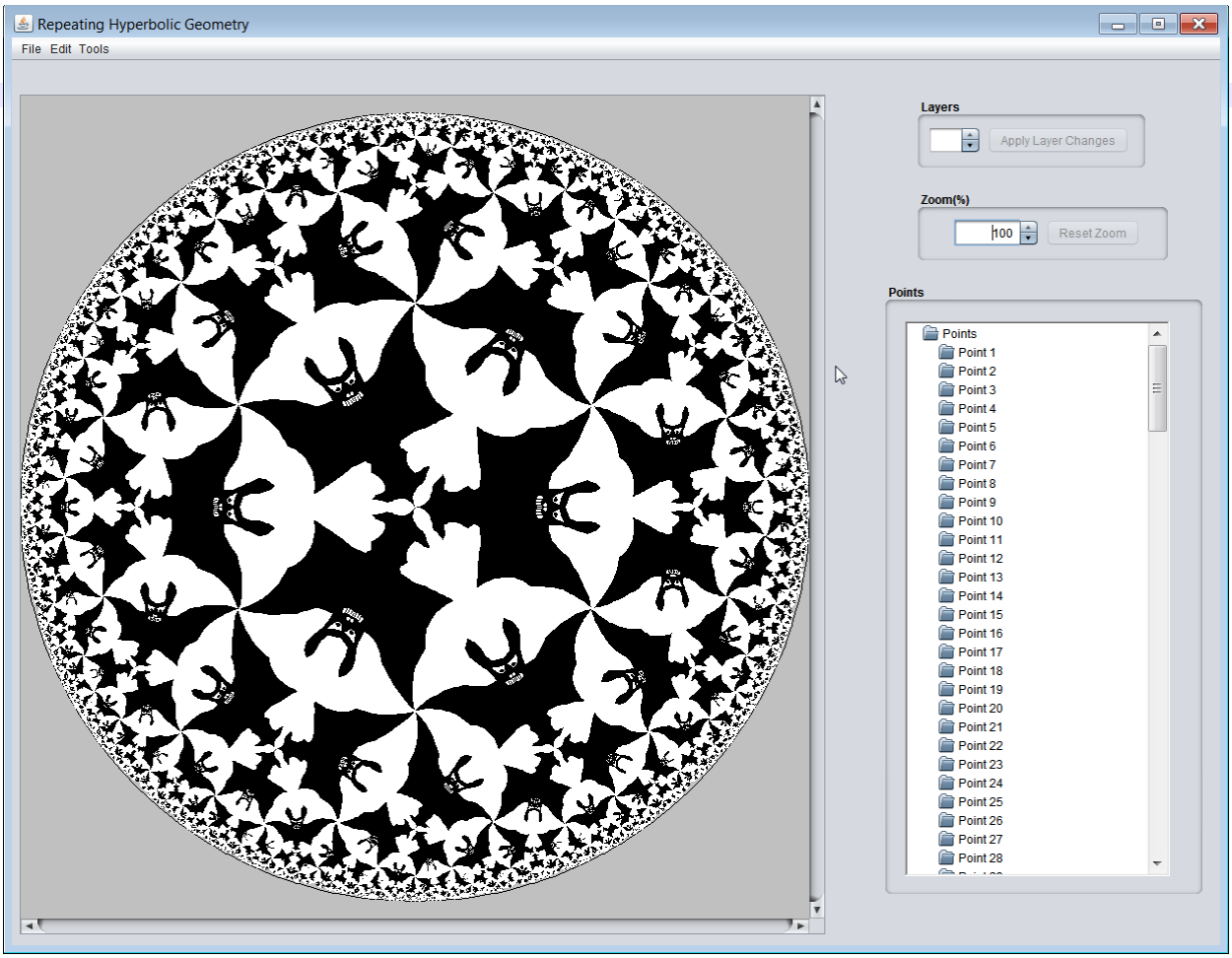

Figure 6.4 Escher's Circle Limit IV pattern (with 4 layers drawn).

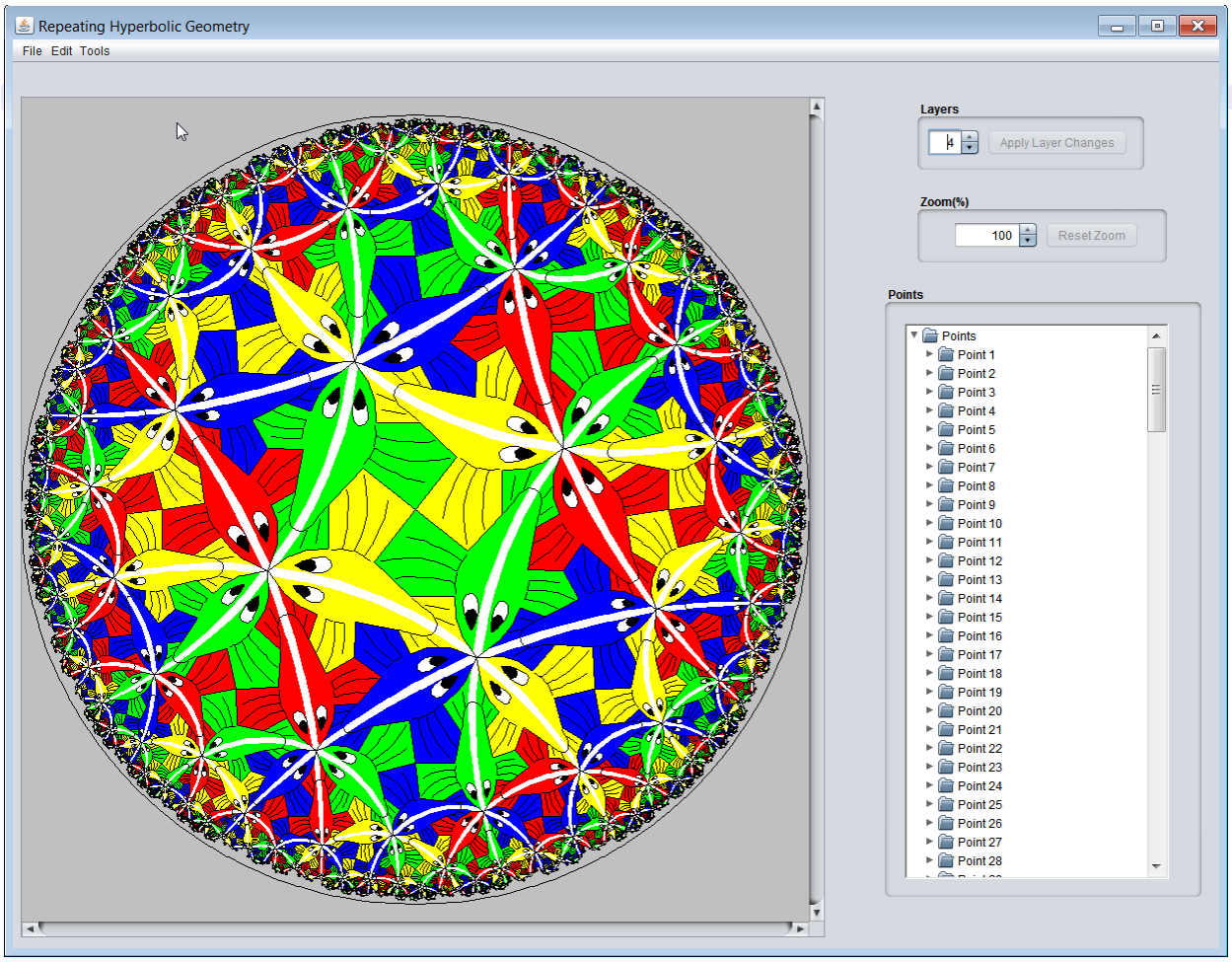

Figure 6.5 Escher's Circle Limit III pattern (with 4 layers drawn).

Figure 6.6 Escher's Circle Limit II pattern (with 4 layers drawn).

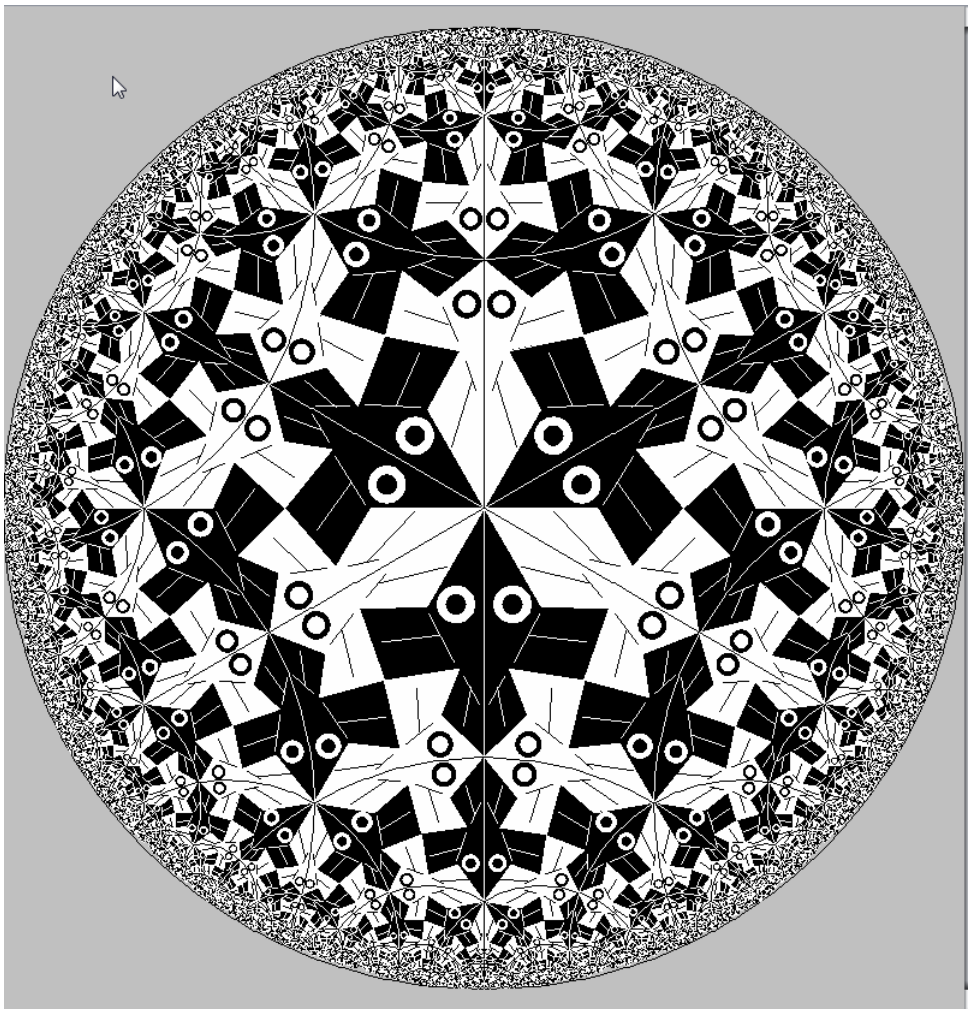

Figure 6.7 Escher's Circle Limit I pattern (with 4 layers drawn).

Chapter 7

Conclusion

This research created a program that generates repeating hyperbolic patterns based on the regular tessellations of the hyperbolic plane, {p, q}, where “p” denotes a regular polygon meeting “q” other regular polygons at each vertex. The program was implemented as a Java applet, which makes the program portable and compatible across platforms and also memory efficient. The applet takes the information from existing data files and is also able to create new patterns using the User Interface of the applet. The use of an applet made the program run and draw the patterns faster than the existing programs. Poincaré’s model of Hyperbolic Geometry was used as the basis for the representation of these repeating patterns. Some of the works of the Dutch artist M. C. Escher, which are considered to be the finest works of art hyperbolic Geometry, have been computerized using this program.

Chapter 8

Future Work

This part of the documentation discusses the improvements and the future work that can be applied to this program of replicating hyperbolic patterns. The user interface of the program can be improvised to have an Interactive way of entering points by clicking and drawing. The capability of using other graphics objects, notably spline curves, could be added to the program. The program could also be extended to allow p or q to be infinity (i.e. to have an infinite number of edges or have all vertices on the bounding circle). More generally, the program could be extended to work with non-regular polygons as the fundamental region.

Drawing these patterns in spherical geometry using Java Applet could be another direction to take. More work could be done in showing these geometric patterns using different models such as the Klein Model or the Weierstrass model. Also research could be done in representing hyperbolic patterns in 3D.

References

[1] Megan Hoopes-Myers. *Tour of tessellations*. http://edtech2.boisestate.edu/meganhoopesmyers/502/virtualtour/history.html, 2010. Accessed 6-5-2013.

[2] Abhijit Parsekar. *A Unified data representation and Visualization of Patterns based on regular tessellations in the three classical geometries*. Master's Thesis, university of Minnesota Duluth 2002.

[3] D. Dunham. *Artistic patterns in hyperbolic geometry*. In Bridges 1999 Conference
Proceedings, pages 239-249, 1999.

[4] D. Dunham. *Creating regular repeating hyperbolic patterns*. In Fifth Mathematics and Design Conference Proceedings, pages 15-20, 2007.

[5] Christopher D. Becker. *Creating Repeating Hyperbolic Patterns Based on Regular Tessellations. Master's Thesis*, university of Minnesota Duluth 2012.

[6] Luis Ollrtegui. *Different kinds of Geometry*. http://www.ehow.com/info_8774739_different-kinds-geometry.html Accessed 6-5-2013.

[7] Marvin Jay Greenberg. *Euclidean and Non-Euclidean Geometries Second Edition*.
W.H. Freeman and Company, New York, 1980.

[8] Joel Castellanos. *What is Non-Euclidean Geometry*. http://www.cs.unm.edu/~joel/NonEuclid/noneuclidean.html. Accessed 6-5-2013.

[9] Faber Richard L. (1983). *Foundations of Euclidean and Non-Euclidean Geometry*. Marcel Dekker Inc, New York.

[10] Dunham, D. 1986, Hyperbolic symmetry, *Computers and Mathematics with Applications*, Volume 12B, pages 139-153.

Appendix A

Data File Format

Here is a sample data file, “p5344.dat"

8 4 2 0 8 1

1 2 3 4 5 6 7 8

1 2 3 4 5 6 7 8

2 1 2 3 4 5 6 7 8

1 1 2 3 4 5 6 7 8

2 1 2 3 4 5 6 7 8

1 1 2 3 4 5 6 7 8

2 1 2 3 4 5 6 7 8

1 1 2 3 4 5 6 7 8

2 1 2 3 4 5 6 7 8

1 1 2 3 4 5 6 7 8

189

0.000000e+00 0.000000e+00 1 4 3

1.116681e-01 9.161260e-02 1 5 3

1.584010e-01 9.135443e-02 1 5 3

1.644652e-01 7.480382e-02 1 5 3

1.827964e-01 3.457978e-02 1 5 3

1.758185e-01 2.143815e-02 1 5 3

2.568180e-01 3.541283e-02 1 5 3

2.382373e-01 5.506387e-02 1 5 3

2.237684e-01 5.742070e-02 1 5 3

2.102659e-01 9.534215e-02 1 5 3

2.322948e-01 1.292710e-01 1 5 3

2.039432e-01 1.200816e-01 1 5 3

1.793350e-01 1.229312e-01 1 5 3

1.670739e-01 1.308884e-01 1 5 3

2.614137e-01 2.136273e-01 1 5 3

2.768638e-01 2.145185e-01 1 5 3

2.853573e-01 1.772605e-01 1 5 3

2.826001e-01 1.311940e-01 1 5 3

3.025063e-01 7.744356e-02 1 5 3

3.396889e-01 7.235643e-02 1 5 3

3.840275e-01 7.536183e-02 1 5 3

4.514057e-01 1.434239e-01 1 5 3

5.946035e-01 2.462928e-01 1 5 3

6.441709e-01 1.143825e-01 1 5 3

6.756667e-01 1.111758e-01 1 5 3

6.847954e-01 1.186661e-01 1 5 3

7.319404e-01 8.537651e-02 1 5 3

7.001589e-01 8.305760e-02 1 5 3

6.691833e-01 6.940678e-02 1 5 3

6.405848e-01 4.619950e-02 1 5 3

6.148748e-01 6.940475e-02 1 5 3

5.795008e-01 1.091672e-01 1 5 3

5.188934e-01 8.836657e-02 1 5 3

4.982978e-01 3.105847e-02 1 5 3

4.447164e-01 0.000000e+00 1 5 3

4.167483e-01 0.000000e+00 1 5 3

3.040999e-01 0.000000e+00 1 5 3

1.993943e-01 0.000000e+00 1 5 3

9.880076e-02 0.000000e+00 1 5 3

0.000000e+00 0.000000e+00 1 6 3

0.000000e+00 0.000000e+00 3 4 3

1.116681e-01 9.161260e-02 3 5 3

1.584010e-01 9.135443e-02 3 5 3

1.644652e-01 7.480382e-02 3 5 3

1.827964e-01 3.457978e-02 3 5 3

1.758185e-01 2.143815e-02 3 5 3

2.568180e-01 3.541283e-02 3 5 3

2.382373e-01 5.506387e-02 3 5 3

2.237684e-01 5.742070e-02 3 5 3

2.102659e-01 9.534215e-02 3 5 3

2.322948e-01 1.292710e-01 3 5 3

2.039432e-01 1.200816e-01 3 5 3

1.793350e-01 1.229312e-01 3 5 3

1.670739e-01 1.308884e-01 3 5 3

2.614137e-01 2.136273e-01 3 5 3

2.768638e-01 2.145185e-01 3 5 3

2.853573e-01 1.772605e-01 3 5 3

2.826001e-01 1.311940e-01 3 5 3

3.025063e-01 7.744356e-02 3 5 3

3.396889e-01 7.235643e-02 3 5 3

3.840275e-01 7.536183e-02 3 5 3

4.514057e-01 1.434239e-01 3 5 3

5.946035e-01 2.462928e-01 3 5 3

4.342015e-01 2.181634e-01 3 5 3

4.140338e-01 1.781443e-01 3 5 3

4.201259128201726e-01 1.612444501314690e-01 3 5 3

3.375083300911032e-01 1.063781884061285e-01 3 5 3

3.576458e-01 1.599189e-01 3 5 3

3.613816e-01 2.136659e-01 3 5 3

3.554259e-01 2.661445e-01 3 5 3

3.990050e-01 2.775246e-01 3 5 3

4.587747e-01 2.920938e-01 3 5 3

4.680294e-01 3.547923e-01 3 5 3

4.368453e-01 3.988088e-01 3 5 3

4.474822e-01 4.474823e-01 3 5 3

2.946856e-01 2.946856e-01 3 5 3

2.150312e-01 2.150312e-01 3 5 3

1.409931e-01 1.409931e-01 3 5 3

6.986269e-02 6.986269e-02 3 5 3

0.000000e+00 0.000000e+00 3 6 3

1.773527e-01 2.049649e-02 3 9 3

1.795605e-01 0.000000e+00 3 11 3

1.735313e-01 4.217946e-02 3 9 3

1.681066e-01 2.810974e-02 3 10 3

1.524500e-01 1.578592e-02 3 10 3

1.281119e-01 1.116550e-02 3 10 3

1.082239e-01 1.157843e-02 3 10 3

1.106118e-01 2.136552e-02 3 10 3

1.005197e-01 2.688620e-02 3 10 3

8.685082e-02 1.968798e-02 3 10 3

7.026504e-02 0.000000e+00 3 11 3

9.492107e-02 2.671279e-02 3 9 3

9.856107e-02 4.043325e-02 3 10 3

9.780687e-02 5.758862e-02 3 10 3

9.688126e-02 6.376341e-02 3 10 3

9.051082e-02 6.773544e-02 3 10 3

8.083014e-02 6.586615e-02 3 11 3

7.313863e-02 4.415221e-02 3 9 3

4.961060e-02 2.724630e-02 3 10 3

3.616276e-02 1.344159e-02 3 10 3

3.392456e-02 0.000000e+00 3 11 3

1.502376e-01 6.653320e-02 3 9 3

1.496574e-01 5.014932e-02 3 10 3

1.385213e-01 3.693012e-02 3 10 3

1.271954e-01 3.262059e-02 3 10 3

1.185362e-01 3.525987e-02 3 10 3

1.251042e-01 5.017778e-02 3 10 3

1.353185e-01 6.461099e-02 3 10 3

1.502376e-01 6.653320e-02 3 11 3

3.121565e-01 1.006290e-02 3 9 3

3.112283e-01 8.791123e-07 3 11 3

2.558182e-01 9.960308e-03 3 9 3

2.544647e-01 8.701251e-07 3 11 3

2.157668e-01 9.901864e-03 3 9 3

2.144372e-01 8.650550e-07 3 11 3

2.819048e-01 1.000472e-02 3 9 3

2.805325e-01 8.739793e-07 3 11 3

3.599890e-01 1.017063e-02 3 9 3

3.590281e-01 8.884846e-07 3 11 3

3.334670e-01 1.010842e-02 3 9 3

3.320484e-01 8.829802e-07 3 11 3

3.866184e-01 1.088233e-02 3 9 3

3.851384e-01 8.943714e-07 3 11 3

4.124008e-01 1.128408e-02 3 9 3

4.103728e-01 9.006177e-07 3 11 3

1.987525e-01 1.585640e-01 3 1 3

2.258972e-01 1.499630e-01 3 2 3

2.251206e-01 1.817563e-01 3 1 3

2.465872e-01 1.686059e-01 3 2 3

2.486436e-01 2.024020e-01 3 1 3

2.627806e-01 1.830102e-01 3 2 3

6.818618e-01 1.005434e-01 3 1 3

7.008116e-01 1.068543e-01 3 2 3

6.915691e-01 9.170244e-02 3 1 3

7.162831e-01 9.587179e-02 3 2 3

3.417625e-01 3.234530e-01 1 9 3

3.852654e-01 3.777304e-01 1 10 3

3.750804e-01 3.556946e-01 1 10 3

3.872492e-01 3.486368e-01 1 10 3

4.222366e-01 3.803186e-01 1 10 3

4.400966e-01 4.147885e-01 1 11 3

2.937257e-01 2.522821e-01 1 9 3

3.257048e-01 2.683460e-01 1 10 3

3.490621e-01 2.879649e-01 1 10 3

3.567427e-01 3.063704e-01 1 10 3

3.282648e-01 2.969674e-01 1 10 3

2.971754e-01 2.749245e-01 1 10 3

2.847236e-01 2.593233e-01 1 10 3

2.937257e-01 2.522821e-01 1 11 3

3.330711e-01 2.841050e-01 1 12 3

3.368285e-01 2.908737e-01 1 13 3

3.290260e-01 2.902909e-01 1 13 3

3.085037e-01 2.745303e-01 1 13 3

2.990212e-01 2.622583e-01 1 13 3

3.124991e-01 2.684784e-01 1 13 3

3.347053e-01 2.843192e-01 1 14 3

4.132894e-01 3.669459e-01 1 9 3

3.997951e-01 3.285513e-01 1 10 3

4.055800e-01 3.005023e-01 1 10 3

4.289473e-01 3.066885e-01 1 10 3

4.595828e-01 3.403340e-01 1 10 3

4.633645e-01 3.608052e-01 1 11 3

4.431983e-01 3.506235e-01 1 9 3

4.461584e-01 3.860980e-01 1 11 3

3.555692e-01 2.657966e-01 1 9 3

3.045616e-01 2.411353e-01 1 10 3

2.738945e-01 2.280235e-01 1 10 3

2.470239e-01 2.239837e-01 1 10 3

2.310804e-01 2.302622e-01 1 11 3

2.738849e-01 2.271613e-01 1 9 3

2.673470e-01 2.501144e-01 1 10 3

2.780514e-01 2.717702e-01 1 10 3

3.418313e-01 3.235321e-01 1 10 3

3.658229e-01 3.310697e-01 1 10 3

3.773368e-01 3.241831e-01 1 11 3

2.424129e-01 2.257366e-01 1 7 3

2.908416e-02 2.662691e-02 1 7 3

2.014244e-01 5.236042e-02 1 1 3

2.172440e-01 2.873257e-02 1 2 3

1.885155e-01 5.007950e-02 1 1 3

1.970986e-01 2.517387e-02 1 2 3

2.140634e-01 5.441955e-02 1 1 3

2.373909e-01 3.208405e-02 1 2 3

4.667724e-01 1.880684e-01 1 1 3

5.084028e-01 2.309322e-01 1 2 3

4.412029e-01 1.821991e-01 1 1 3

4.661780e-01 2.233911e-01 1 2 3

4.971783e-01 1.974719e-01 1 1 3

5.448714e-01 2.373884e-01 1 2 3

The pattern corresponding to the above data file is shown below:

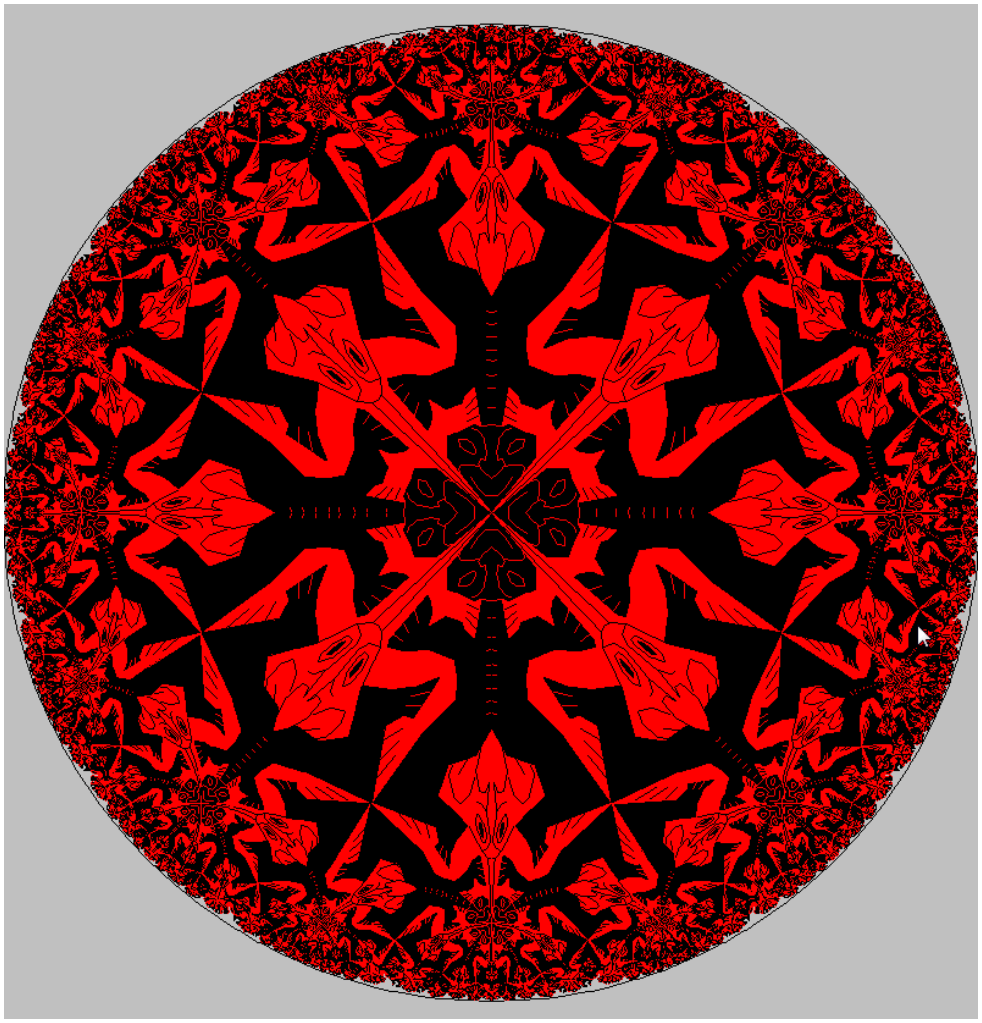

Figure A.1: A Pattern Based on {8, 4} (With 4 layers drawn)

In the first line, 8 4 2 0 8 1:

- The first number is the value of p, i.e. p = 8 in this case. The central polygon in

  Figure A1 is an 8-gon.

- The second number is the value of q, i.e. q = 4 in this case (this pattern is based on the tessellation {8, 4}). The 8-gon in Figure A.1 meets three other 8-gons at each vertex.

- The third number, 2 in this case, is the number of “different" sides of the central p-gon that are used to form the fundamental region (the other sides of the fundamental region are two radii from the center to two vertices of the central p-gon separated by 2 (2 = p)). This number must divide p, and p divided by this number is the number of copies of the motif that appears in the central p-gon.

- The fourth number is not used and is there merely to maintain compatibility with older versions of the program.

- The fifth number, 8 in this case, must be the highest “color" number of the colors used. The color numbers are:

    1. 1 Black
    2. 2 White
    3. 3 Red
    4. 4 Green
    5. 5 Blue
    6. 6 Cyan
    7. 7 Magenta
    8. 8 Yellow

9. 9 Salmon
10. 10 Brown

- The sixth number, 1 in this case, indicates the kind of reflection symmetry the pattern has within the central p-sided polygon:
    1. 0 indicates that there is no reflection symmetry (only rotation symmetry).
    2. 1 indicates that there is reflection symmetry across the perpendicular bisector of one of the edges of the p-gon.
    3. 2 indicates that there is reflection symmetry across a radius (from the center to a vertex of the p-sided polygon).

The second line, 1 2 3 4 5 6 7 8 is the color permutation induced by rotating by 2 (2 = p) (i.e., the third number of line 1 times 2 (2 = p). Note that this is the “array" representation of permutations (not the \mathematical" one using cycles): the values listed are the values of perm [1], perm [2], etc.

The third line, 1 2 3 4 5 6 7 8, is the color permutation induced by the reflection, if the sixth number of line 1 is 1 or 2 (it is just the identity, if the sixth number is 0).

The next p lines consist of a first number followed by a color permutation. The first number of the first of these lines indicates which edge (edge 2 in this case) of the transformed p-gon should lie next to edge 1 of the central p-gon. In general, if this first number is positive, the transformed p-gon is rotated into position; if the number is negative, a reflection is used to move the transformed p-gon into position. Note

that the edges are numbered from 1 to p, not from 0 to p-1, so that the edges can be assigned an unambiguous sign (i.e. 0 is not used as +0 = -0). The next eight numbers, 1 2 4 8 5 6 7 3 (perm [1] = 1, perm [2] = 2, perm [3] = 4, etc.), define the color permutation that will be induced when we go across this edge. The initial color permutation is always assumed to be the identity permutation.

The first number of the second of these lines indicates which edge (edge 1 in this case) of the transformed p-gon should lie next to edge 2 of the central p-gon. In this case, the color permutation is 1 2 3 4 5 6 7 8. This pattern continues for six more lines.

The next line consists of a single number, the number of points that make up the motif. It is 189 in this case. Following that line are 189 lines of numbers each; each line species one point. Each line has the following format: x-coordinate y-coordinate color point-type number-of-layers.

- The x-coordinate and y-coordinate are within the central p-gon (and hence the unit circle).
- The color is one of the color numbers discussed previously.
- The point-type is one of:
    1. 1 “Move To"
    2. 2 “Draw To"
    3. 3 “Circle" (there must be two of these in succession)
    4. 4 Start a (Euclidean) “Filled Polygon"
    5. 5 Continue a (Euclidean) “Filled Polygon"

6. 6 End a (Euclidean) “Filled Polygon"
7. 7 “Hyperline" (there must be two of these in succession)
8. 8 “Filled Circle" (there must be two of these in succession)
9. 9 Start a (Euclidean) “Polyline"
10. 10 Continue a (Euclidean) “Polyline"
11. 11 End a (Euclidean) “Polyline"
12. 12 Start a (hyperbolic) “Filled p-gon"
13. Continue a (hyperbolic) “Filled p-gon"
14. End a (hyperbolic) “Filled p-gon"

- The number-of-layers is not used and is there merely to maintain compatibility with older versions of the program.

Appendix B

Software Architecture

Classes

Here are the classes with brief descriptions:

DimensionMismatchException: An exception thrown when the dimensions of input do not meet requirements.

InvalidIndexValueException: An exception thrown when the index used is not valid.

InvalidPointTypeException: An exception thrown when a Motif Point Type is not valid for the surrounding types.

MalformedDataFileException: An exception thrown when an invalid or malformed data is read from the input file.

Colorenum: Implementation file for the *DefinedColor* enum utility functions. Enum DefinedColor is defined as follows:

```
enum DefinedColor
{
COLOR_GREY,
COLOR_BLACK,
COLOR_WHITE,
COLOR_RED,
```

COLOR_GREEN,

COLOR_BLUE,

COLOR_CYAN,

COLOR_MAGENTA,

COLOR_YELLOW,

COLOR_SALMON,

COLOR_BROWN;

}

Functions:

- static DefinedColor *getColorFromInt*(int color) throws InvalidIndexValueException - A function that will be called when converting from integer to DefinedColor enum value.
- static Color *getColorFromDefinedColor*(DefinedColor color) throws InvalidIndexValueException - A function that will be called when converting from the DefinedColor enum to colors used by java.
- static int *getIntFromDefinedColor*(DefinedColor color) - A function that converts a DefinedColor to integer value.
- Color *colorToString*(DefinedColor color) - A function that converts a DefinedColor to a string.

DrawingUtility: Class to hold the Drawing Helper Functions.

Functions:

- static VectorList *hyperbolicLine*(final VectorList startPoint, final VectorList endPoint) throws InvalidIndexValueException - A utility function to help calculate hyperbolic lines.
- static VectorList *resizeHyperbolicPoint*(final VectorList point, int sceneSize)throws InvalidIndexValueException - A utility function to resize a hyperblic point to the Java Applet coordinate system.

EditFileDialog: A class that allows the user to create or edit the basic aspects of a motif using a dialog box

Functions:

- public final void *initUI*() - A function that designs the dialog box and its UI.
- protected void *showEvent*() - A function that initiates the dialog box.
- void *addTransformationDataWidgets*() - A utility function to add the dynamically allocated Widgets for the transformation data.
- void *changeRotateAndReflectColorWidgets*() - A utility function to change the color permutations for rotate and reflect colors.
- void *addRotateAndReflectColorWidgets*() - A utility function to add the dynamically allocated Widgets for color permutations for rotate and reflect colors.
- void *accept*() - A function that does the processing after the values in the dialog box are given.

- private void *changeP*() - A utility function to handle changes needed when p changes.
- void *removeRotateAndReflectColorWidgets*() - A utility function to remove the dynamically allocated Widgets for color permutations for rotate and reflect colors.
- void *removeTransformationDataWidgets*() - A utility function to remove the dynamically allocated Widgets for the transformation data.
- int *getNumberOfColors*() - Gets the number of colors for the new/changed motif.
- int *getP*() - Gets p for the new/changed motif.
- int *getQ*() - Gets q for the new/changed motif.
- Vector<Integer> *getReflectionColors*() - Gets the reflection colors for the new/changed motif.
- int *getReflectionType*() - Gets the reflection type for the new/changed motif.
- int *getReflectionType*() - Gets the reflection type for the new/changed motif.
- Vector<Integer> *getRotationColors*() - Gets the rotation colors for the new/changed motif.
- int *getSidesInCentralRegion*() - Gets the number of sides in the central region for the new/changed motif.
- Vector<Vector<Integer> > *getTransformationInformation*() - Gets the Transformation information for the new/changed motif.
- void *setNumberOfColors*(int n) - Sets the number of colors for the new/changed motif.

- void *setP*(int p) - Sets p for the new/changed motif.
- void *setQ*(int q) - Sets q for the new/changed motif.
- void *setTransformationInformation*(Vector<Vector<Integer> > t) - Sets the transformation information for the new/changed motif.
- void *clean*() - A utility function to clean up dynamically allocated Widgets.
- void *close*() - Closes the dialog box.
- void *setSidesInCentralRegion*(int s) - Sets the number of sides in the central region for the new/changed motif.
- void *setRotationColors*(Vector<Integer> r) - Sets the rotation colors for the new/changed motif.
- void *setReflectionType*(int reflectionType) - Sets the reflection type for the new/changed motif.
- void *setReflectionColors*(Vector<Integer> r) - Sets the reflection colors for the new/changed motif.

EditPointDialog: A class that allows the user to create or edit a point (or group of points.

Functions:

- public final void *initUI*(int color, ObjectType obj, int max, int min) - A function that designs the dialog box and its UI.
- public void *accept*() - A function that does the processing after the values in the dialog box are given.

- public void *showEvent*() throws InvalidIndexValueException - A function that initiates the dialog box.
- private void *removeLastPoint*() - A utility function to remove the last point in the list.
- private void *addPoint*(double x, double y) - A utility function to add a point to the list with coordinate (x, y).
- private void *addPointPushed*() - A function called when the Add Point Button is clicked.
- private void *removePointPushed*() - A function called when the Remove Point Button is clicked.
- private void *cleanup*() - A utility function to perform various cleanup operations.
- private JSpinner *constructSpinBox*(double value, JSpinner returnValue) - A utility function to construct a spin box.
- public Vector<MotifPoint> *getPoints*() - Returns the points generated after the dialog has been accepted.
- public void *setMaximumColor*(int maxColor) - Sets the maximum color number to use.
- public void *setMaximumPoints*(int maxPoints) - Sets the maximum number of points to use.
- public void *setMinimumPoints*(int minPoints) - Sets the minimum number of points to use.

- public void *setObjectType*(ObjectType type) - Sets the object type for the object being added/edited.
- public void *setPoints*(Vector<MotifPoint> points) - Sets the points to use before displaying the dialog.
- void *close*() - Closes the dialog box.

Extramath: A class for math functions specific to this project.

Functions:

- static double *myCosh2*(int p, int q) - A function to compute the hyperbolic cosine of 2b using p and q.
- static double *mySinh2*(int p, int q) - A function to compute the hyperbolic sine of 2b using p and q.
- static double *myCosh*(int p, int q) - A function to compute the hyperbolic cosine of b using p and q.

GraphicsItem: A class that holds the properties of a GraphicsItem like Shape, Color, etc.

MainWindow: A class that creates the main window for the application.

Functions:

- void *addCircle* () - A function to add a Circle.
- void *addDrawToPoint* () - A function to add a Draw To Point.
- void *addFilledCircle* () - A function to add a Filled Circle.
- void *addFilledPGon* () - A function to add a Filled PGon.

- void *addFilledPolygon* () - A function to add a Filled Polygon.
- void *addMoveToPoint* () - A function to add a Move To Point.
- void *addPolyline* () - A function to add a Polyline.
- void *changeLayers* () - A function to update the number of layers desired by the user.
- void *changeZoom* () - A function to update the zoom level desired by the user.
- void *getSettings* () - A function to get settings from the user.
- void *newFile* () - A function that will be called when the user wants a new file.
- void *openFile* () - A function that will be called when the user wants to open a file.
- void *resetZoom* () - A function to reset the zoom level to 100%.
- void *saveFile* () - A function that will be called when the user wants to save a file.
- void *deleteGroup* () - A function to delete a point group.
- void *editGroup* () - A function to edit a point group.
- void *enableChangeLayers* () - A function to enable the Apply Layer Changes button.
- void *moveGroupDown* () - A function to move a point group down.
- void *moveGroupUp* () - A function to move a point group up.
- void *pointSelected* () - A function to handle when a point is selected in the point list.

- void *closeEvent* ( ) - A function to handle the close event of the MainWindow.
- private void *initComponents*() - Designs the components of the main applet.
- private void *initializeVariables*() throws InvalidIndexValueException, DimensionMismatchException - A utility function to initialize variables.
- void *replicateMotif*(Vector<MotifPoint> passedPoints) throws IOException, InvalidIndexValueException, DimensionMismatchException, InvalidPointTypeException - A utility function to recursively replicate the motif.
- void *recursiveReplicateMotif*(final Vector<MotifPoint> passedPoints, boolean exposure, int currentLayer, int stopLayer, Transformation passedCurrentTransformation) throws DimensionMismatchException, InvalidIndexValueException, IOException, InvalidPointTypeException - A utility function to recursively replicate the motif.
- Vector<GraphicsItem> *generateMotif*(int layer, final Vector<MotifPoint> passedPoints, Matrix Temptransform) throws InvalidIndexValueException, IOException, InvalidPointTypeException - A utility function to create the motif object to be drawn.
- void *updatePointsList*() throws InvalidIndexValueException - A utility function to update the point list widget.

- void *moveTo*(final VectorList point, boolean flag) throws InvalidIndexValueException, IOException - A utility function to help move to a point.
- Vector<MotifPoint> *generatePGonPattern*(Vector<MotifPoint>tmpoints) throws InvalidIndexValueException, DimensionMismatchException - A utility function to fill in a layer.
- Transformation *addToTransformation*(final Transformation transformation, int shift) throws DimensionMismatchException, InvalidIndexValueException - A utility function to add to a transformation.
- Transformation *computeTransformation*(final Transformation transformation, int shift) throws DimensionMismatchException, InvalidIndexValueException - A utility function to compute a Transformation.
- Transformation *createTransformation*(final int i, int orientation, final Vector<Integer> colorPermutation) throws DimensionMismatchException, InvalidIndexValueException - A utility function to create a transformation.
- void *determineItemGroup*(int itemIndex,int startIndex,int endIndex) throws InvalidIndexValueException - A utility function to find the point index range for point group given one item in the group.
- public boolean *isEvenBefore*(int startIndex, PointType type) - A utility function to determine if the number of points of a given type immediately before an index is even.
- int *findEnd*(int startIndex, PointType type) - A utility function to find the end of a sequence.

- int *findStart*(int startIndex, PointType type) - A utility function to find the start of a sequence.
- GraphicsItem *generateLineTo*(final VectorList point, Color color, boolean flag) - A utility function to help generate a line to a point from previous call to moveTo.
- GraphicsItem *generateHyperbolicLineCurve*(double x1, double y1, double x2, double y2, Color color, boolean flag) throws InvalidIndexValueException, IOException - A utility function to generate a hyperline curve.
- GraphicsItem *generateHyperbolicLineCurve*(final VectorList point1,final VectorList point2,Color color,boolean flag) throws InvalidIndexValueException, IOException - A utility function to generate a hyperline curve.
- GraphicsItem *generateCircle*(final VectorList point1, final VectorList point2, Color color, boolean flag) throws InvalidIndexValueException, IOException - A utility function to help generate a circle.
- GraphicsItem *generateFilledCircle*(final VectorList point1, final VectorList point2,Color color,boolean flag) throws InvalidIndexValueException, IOException - A utility function to help generate a filled circle.
- GraphicsItem *generateFilledPgon*(final Vector<VectorList> points, Color color, boolean flag) throws InvalidIndexValueException, IOException - A utility function to help generate a filled pgon.

- GraphicsItem *generateFilledPolygon*(final Vector<VectorList> passedPoints, Color color, boolean flag) throws InvalidIndexValueException, IOException - A utility function to help generate a filled polygon.
- GraphicsItem *generatePolyline*(final Vector<VectorList>passedPoints, Color color, boolean flag) throws InvalidIndexValueException, IOException - A utility function to help generate a polyline.
- void *drawLineTo*(final VectorList point, Color color, boolean flag)throws InvalidIndexValueException, IOException - A utility function to help draw a line to a point from previous call to moveTo.
- void *drawHyperbolicLineCurve*(double x1, double y1, double x2, double y2, Color color, boolean flag) throws InvalidIndexValueException, IOException - A utility function to draw a hyperline curve.
- void *writeMotif*(final String fileName, final int p, final int q, final int sidesInCentralRegion, final int numberOfColors, final int reflectionType, final Vector<Integer> rotationColors, final Vector<Integer> reflectionColors, final Vector<Transformation> edgeTransformations, final Vector<MotifPoint> points) throws IOException - A utility function to write a motif file.

Transformation: Class to represent a transformation.

Functions:

- public Transformation *times*(Transformation right) throws DimensionMismatchException, InvalidIndexValueException - Implementation of the * operator.
- public Transformation assign(Transformation t) - Implementation of the = operator.

Utility: Class to hold the General Helper Functions.

Functions:

- Vector< double > *destereofyPoint* (double x, double y, double z) - A utility function to translate from 3D point to 2D point.
- Vector< double > *destereofyPoint* (const Vector< double > &xyz) throw (InvalidIndexValueException) - A utility function to translate from 3D point to 2D point.
- Vector< double > *stereofyPoint* (double x, double y) - A utility function to translate from 2D point to 3D point.
- Vector< double > *stereofyPoint* (const Vector< double > &xy) throw (InvalidIndexValueException) - A utility function to translate from 2D point to 3D point.
- Vector< double > *translateKleinToPoincare* (double x, double y) - A utility function to translate from Klein model to Poincare model.

- Vector<double>*translateKleinToPoincare* (const Vector<double>&xy) throw (InvalidIndexValueException) - A utility function to translate from Klein model to Poincare model.
- Vector< double > *translatePoincareToKlein* (double u, double v) - A utility function to translate from Poincare model to Klein model.
- Vector<double>*translatePoincareToKlein* (const Vector<double>&uv) throw (InvalidIndexValueException) - A utility function to translate from Poincare model to Klein model.
- vector< MotifPoint > *verifyAndArrangePoints* (vector< MotifPoint > points) - A utility function to create the motif.

MotifPoint: A class that holds the properties of a motif point.

MyDrawingPanel: A class that draws the shapes on the canvas using paintComponent.

ObjectTypeEnum: Implementation file for the *ObjectType* enum utility functions. Enum ObjectType is defined as follows:

enum ObjectType

{

OBJECT_TYPE_MOVE_TO (1),

OBJECT_TYPE_DRAW_TO (2),

OBJECT_TYPE_CIRCLE (3),

OBJECT_TYPE_FILLED_POLYGON (4),

OBJECT_TYPE_HYPERLINE (7),

OBJECT_TYPE_FILLED_CIRCLE (8),

OBJECT_TYPE_POLYLINE (9),

OBJECT_TYPE_FILLED_PGON (12);

}

PointTypeNum: Implementation file for the *PointType* enum utility functions.

Enum PointType is defined as follows:

```
enum PointType
{
   POINT_TYPE_MOVE_TO (1),
   POINT_TYPE_DRAW_TO (2),
   POINT_TYPE_CIRCLE (3),
   POINT_TYPE_START_FILLED_POLYGON (4),
   POINT_TYPE_CONTINUE_FILLED_POLYGON (5),
   POINT_TYPE_END_FILLED_POLYGON (6),
   POINT_TYPE_HYPERLINE (7),
   POINT_TYPE_FILLED_CIRCLE (8),
   POINT_TYPE_START_POLYLINE (9),
   POINT_TYPE_CONTINUE_POLYLINE (10),
   POINT_TYPE_END_POLYLINE (11),
   POINT_TYPE_START_FILLED_PGON (12),
   POINT_TYPE_CONTINUE_FILLED_PGON (13),
   POINT_TYPE_END_FILLED_PGON (14);
```

}

UserModeEnum: Implementation file for the *UserMode* enum utility functions.

Enum UserMode is defined as follows:

enum UserMode { READ_ONLY, EDIT };