5-2012

Manipulation of 3D knotted polygons

Sairaj Rachamadugu
Western Kentucky University, sairaj.rachamadugu360@topper.wku.edu

MANIPULATION OF 3D KNOTTED POLYGONS

A Thesis
Presented to
The Faculty of the Department of Mathematics and Computer Science
Western Kentucky University
Bowling Green, Kentucky

In Partial Fulfillment
Of the Requirements for the Degree
Master of Science

By
Sairaj Rachamadugu

May 2012

MANIPULATION OF 3D KNOTTED POLYGONS

Date Recommended 4/26/2012

[illegible]

Dr. Uta Ziegler, Director of Thesis

[illegible]

Dr. Claus Ernst

[illegible]

Dr. Guangming Xing

[illegible] 21-May-2012

Dean, Graduate Studies and Research Date

ACKNOWLEDGMENTS

First and foremost, I give all credit to God for the strength and knowledge to complete this project. Without him, none of this would have been possible and it is all for his glory.

There are number of individuals who contributed to the completion of this project. I would like to express my most sincere thanks to the members of my committee, Dr. Uta Ziegler, Dr.Claus Ernst and Dr.Guangming Xing.

I would like to recognize my thesis director Dr.Uta Ziegler, for her wonderful guidance and also for the vast amount of time she spent in reading and critiquing many drafts of this thesis. Under her supervision, I have learned a great deal about the research process. I am extremely proud to have such a dedicated professor as my mentor and friend. I would like to thank Dr.Claus Ernst for his professional insight and moral support. I am thankful that I have had the opportunity to work closely with one of the finest human beings. I would like to thank Dr.Guangming Xing for his guidance throughout my masters and also for his motivative support. I especially thank him for providing an opportunity as a graduate assistant in the year 2011, which eventually turned my life in a positive way.

Finally, I would specially thank my mother, R. Neeraja, my father, R. Manohar Rao, my brother, Sai krishna and also my sweet sister, Tejaswini Pothuri, who were always there when I needed the moral support and encouragement. I would also like to thank my friends, Srujan Gopu and Rajesh Varkur who gave me valuable advice. My sincere thanks to all who have helped me accomplish this project.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ALGORITHMS

MANIPULATION OF 3D KNOTTED POLYGONS

Sairaj Rachamadugu May 2012 59 Pages

Directed by Dr.Uta Ziegler, Dr.Claus Ernst, and Dr.Guangming Xing

Dept. of Mathematics and Computer Science Western Kentucky University

This thesis discusses the development of software architecture to support the computational investigation of random polygons in 3 space. The random polygons themselves are a simple model of long polymer chains. (A DNA molecule is one example of a polymer.)

This software architecture includes “building blocks” which specify the actual manipulations and computations to be performed, and a structural framework which allows the user to specify which manipulations/computations to perform, in which order and with how many repetitions. The overall framework is designed in such a way that new building blocks can easily be added in the future. The development of three different building blocks to be used in this architecture which are entitled: Reducer, Lengthener and OutsideInLengthener are also discussed in this thesis. These building blocks manipulate the existing polygons - increasing or decreasing their size.

Chapter 1

INTRODUCTION

Who do you think are the key players in the complex network interaction that takes place in every organism? They are the macromolecular self-assembly processes. One such self-assembly process is the packing of the genetic material in the capsids of viruses, such as the double-stranded DNA molecule chain packed into the capsid of bacteriophage P4. From the experimental data we can draw a conclusion that the two ends of the DNA strand which meet in the capsid of a P4 virus mutant create a circular molecule and a high percentage of knots are a result of this circularization reaction [12]. However, direct observation of the process or results is difficult, which implies that the assembly process is not well understood. So, a computational investigation may be helpful in getting an insight.

In this thesis we model the DNA in a viruses capsid by considering circular molecules confined to a small volume as polygons confined in a ball. The main idea of this thesis is to develop a software architecture which supports computational investigation of a DNA model using a simple model i.e., polygons confined by a sphere. The software architecture includes certain modules which manipulate the polygons. It allows the user to specify which manipulations to apply and in which order. This architecture is also designed in such a way that other modules can be easily added in the future. As a part of this thesis, 3 modules are developed which are entitled: Reducer, Lengthener and OutsideInLengthener. These modules should not be considered to work with a model of DNA molecule directly. The molecule can't change its length or flexibility, but a polygon can become larger/shorter or vice versa by adding/reducing the number of vertices.

The Reducer is one polygon simplification algorithm which includes geometrical manipulations that preserve the knot type. It helps in reducing the polygon into a more manageable form while preserving the knot type.

Lengthener and OutsideInLengthener are two different approaches which help in lengthening and relaxing the polygon to an extent.

Using these three modules in sequence might result in a more simplified form of the original polygon because, while performing reducer module there are chances that the polygon may get struck at a certain stage. So, Lengthener and OutsideInLengthener operate to relax the polygon such that successive attempts by the Reducer have a chance to simplify the polygon further.

The rest of this thesis is organized as follows:

Chapter 2 describes the background knowledge of certain basic concepts including knot theory [2], plane geometry [1] and linear algebra [9] and also the common algorithms that are used in the later chapters.

Chapter 3 presents the three kinds of manipulations that are performed on the polygon and corresponding graphic results are shown using mathematica [15].

Chapter 4 describes the framework and its use and also presents the graphical user interface developed, using swings for java [3].

Chapter 5 gives a conclusion and gives future directions of this research.

Chapter 2

BACKGROUND

This section provides important background information about topics related to topology, plane geometry and linear algebra which will be used in later chapters. More detailed information can be found in standard texts [1, 2, 9]. This chapter also introduces common algorithms that are used in the later chapters.

2.1 Basic definitions

2.1.1 Topology

This subsection lists the basic concepts that are used in this proposal with brief definitions.

2.1.1.1 Knot Theory

Knot theory is the mathematical study of knots [2].

2.1.1.2 Knot

A knot is intuitively defined as a simple closed curve in R^3 without self-intersections. Two or more disjointed knots together are called a link. A knot can be deformed, twisted or stretched as long as there occurs no self intersections. If a knot K_1 is transformed into another knot K_2 without self intersections then K_1 and K_2 are said to be topologically equivalent.

2.1.1.3 Knot Type

A knot type can be defined as the set of all equivalent knots.

2.1.1.4 Polygon

A polygon is a 3 dimensional object with n vertices in 3D i.e., v_0, v_1,..., v_{n-1} and with n edges; e_i connects v_i and v_{i+1} for $0 \leq i <$ n-1 and e_{n-1} connects v_{n-1} and v_0.

2.1.2 Linear Algebra

2.1.2.1 Vectors

A vector $\overrightarrow{v}$ in R^n can be referred to by the number of co-ordinates it has, so a 2-dimensional vector $\begin{bmatrix} v_x \\ v_y \end{bmatrix}$ is often called a 2-vector, and 3-dimensional vector $\begin{bmatrix} v_x \\ v_y \\ v_z \end{bmatrix}$ is often called 3-vector [9].

A vector from a point A to a point B is denoted by $\overrightarrow{v} = \overrightarrow{AB} =$ B-A. Point A is called the tail of the vector $\overrightarrow{v}$ and B is called the head of the vector $\overrightarrow{v}$.

A vector has both length and a direction. A vector $\overrightarrow{v}$ is shown in

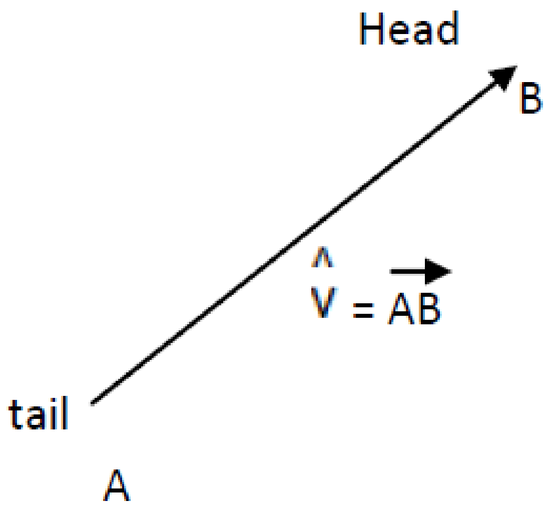


Figure 2.1: Vector $\hat{v}$

Figure 2.1. Two or more vectors can be added together (vector addition), subtracted (vector subtraction) and multiplied by scalars (scalar multiplication). Vector multiplication can be defined for a pair of vertices by dot product and the cross product.

2.1.2.2 Length of a Vector

The length of the 3-Dimensional vector $\overrightarrow{v}$ = {v$_x$, v$_y$, v$_z$} is denoted by $|v|$, and is computed as

$$|v| = \sqrt{v_x^2 + v_y^2 + v_z^2}$$

2.1.2.3 Unit Vector

A unit vector is a vector of length l, sometimes also called a direction vector. The unit vector $\hat{v}$ of a vector $\overrightarrow{v}$, having the same direction as a given (non-zero) vector $\overrightarrow{v}$ is defined by $\hat{v} = \frac{\overrightarrow{v}}{|v|}$, where $|v|$ denotes the norm of vector or the length of vector $\overrightarrow{v}$.

2.1.2.4 Dot product

The dot product of two vectors x and y is denoted by x.y = $|x||y|\cos(\theta)$, where θ is the angle between the vectors x and y. It follows immediately that x.y = 0 iff x is perpendicular to y.

2.1.2.5 Cross product

Let $\hat{x}$ = (1, 0, 0), $\hat{y}$ = (0, 1, 0), $\hat{z}$ = (0, 0, 1) be the standard vectors in R^3. Using these standard vectors, for vectors u = $\begin{bmatrix} u_x \\ u_y \\ u_z \end{bmatrix}$, v = $\begin{bmatrix} v_x \\ v_y \\ v_z \end{bmatrix}$ in R^3, the cross product is denoted by u $\times$ v and is defined to be u $\times$ v = $\hat{x}$ ($u_y v_z$ - $u_z v_y$) - $\hat{y}$ ($u_x v_z$ - $u_z v_x$) + $\hat{z}$ ($u_x v_y$ - $u_y v_x$).

2.1.3 Plane Geometry

2.1.3.1 Coordinate Plane

A two dimensional surface in which points are plotted and located by their x and y coordinates.

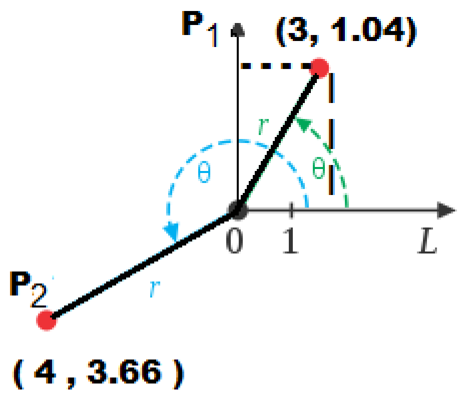


Figure 2.2: Polar coordinates

2.1.3.2 Polar Coordinate System

A point p = {x, y} in a Cartesian coordinate system can be expressed in polar form {d, θ} as shown in Figure 2.2, where d = |p| and θ is the counter clockwise angle between the vectors {1, 0} and {x, y}.

2.1.3.3 Area of Triangle

For given three non collinear vertices A = (x_1, y_1, z_1) , B = (x_2, y_2, z_2) and C = (x_3, y_3, z_3), the area of △ABC is given by

$$\triangle = \frac{1}{2}\left|(B - A) \times (A - C)\right|$$

and the centroid of the same triangle is given by

$$G = \frac{1}{3}(A + B + C)$$

2.1.3.4 Solid Cylinder

A solid cylinder is a solid consisting of two congruent disks in parallel planes called bases and the axis of the cylinder is the line connecting the two center points of the bases and is perpendicular to both bases.

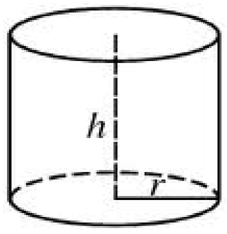


Figure 2.3: Cylinder

A solid cylinder is represented with radius r and height h where r is the radius of the base and h is the length of the axis shown in Figure 2.3.

The rest of this thesis refers to a solid cylinder as a cylinder.

2.1.3.5 Equation of line segment

If $P_1 = (x_1, y_1, z_1)$ and $P_2 = (x_2, y_2, z_2)$ are the two end points of a line segment then the equation of a line segment is given by

$$Q = Q(s) = P_1 + s(P_2 - P_1) \tag{2.1}$$

where $s \epsilon [0, 1]$.

In the above equation the value of s determines the position of the resultant point Q on the line segment. For instance, if the value of $s = 0$ then the point Q is at P_1. Similarly if the value of $s = 1$ then the point Q is at P_2. The point Q moves along the line segment from P_1 to P_2 as the values of s includes in [0, 1].

2.1.3.6 Equation of a plane

The standard equation of a plane denoted as F in R^3 is given by

$$F = F(x, y, z) = Ax + By + Cz + D = 0 \tag{2.2}$$

where A, B, C and D are constants.

Given three points in space (x_1, y_1, z_1), (x_2, y_2, z_2) and (x_3, y_3, z_3) then the equation of the plane through these points is given by the following determinants as

$$A = \begin{vmatrix} 1 & y_1 & z_1 \\ 1 & y_2 & z_2 \\ 1 & y_3 & z_3 \end{vmatrix} \quad B = \begin{vmatrix} x_1 & 1 & z_1 \\ x_2 & 1 & z_2 \\ x_3 & 1 & z_3 \end{vmatrix} \quad C = \begin{vmatrix} x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \\ x_3 & y_3 & 1 \end{vmatrix} \quad D = - \begin{vmatrix} x_1 & y_1 & z_1 \\ x_2 & y_2 & z_2 \\ x_3 & y_3 & z_3 \end{vmatrix}$$

Expanding the above determinants gives

$A = y_1 (z_2 - z_3) + y_2 (z_3 - z_1) + y_3 (z_1 - z_2)$

$B = z_1 (x_2 - x_3) + z_2 (x_3 - x_1) + z_3 (x_1 - x_2)$

$C = x_1 (y_2 - y_3) + x_2 (y_3 - y_1) + x_3 (y_1 - y_2)$

$D = x_1 (y_3 z_2 - y_2 z_3) + x_2 (y_1 z_3 - y_3 z_1) + x_3 (y_2 z_1 - y_1 z_2)$

2.2 Common Algorithms

2.2.1 Point of intersection of plane and the line segment

Given a plane F which is determined by the points R_1, R_2, R_3 and a line segment g which has P_1, P_2 as its end points; then the point of intersection can be determined by solving four simultaneous equations

$Ax + By + Cz + D = 0$

$$Q_x = P_{1_x} + s\,(P_{2_x} - P_{1_x}) \tag{2.3}$$

$$Q_y = P_{1_y} + s\,(P_{2_y} - P_{1_y}) \tag{2.4}$$

$$Q_z = P_{1_z} + s\,(P_{2_z} - P_{1_z}) \tag{2.5}$$

for Q_x, Q_y, Q_z and s, giving

$$s = \frac{AP_{1x}+BP_{1y}+CP_{1z}+D}{A(P_{1x}-P_{2x})+B(P_{1y}-P_{2y})+C(P_{1z}-P_{2z})} \tag{2.6}$$

If the value of s ϵ [0, 1], then substitute it back into equations 2.3, 2.4 and 2.5 to give a point of intersection (Q_x, Q_y, Q_z). In equation 2.6, if the value of denominator is 0 then the line segment is parallel to the plane and there are no solutions.

Algorithm 1 describes the point of intersection between plane and a line segment.

Algorithm 1 Point of intersection of plane and the line segment

```
Input: 3 points $R_1$, $R_2$, $R_3$ which are used to define a
plane F and 2 points $P_1$, $P_2$ which define a line segment.
Output: The point of intersection Q of the line segment
$\overline{P_1P_2}$ with the plane F.
Restrictions: $R_1$, $R_2$, $R_3$ should not be collinear and $\overline{P_1P_2}$
should not be parallel to F.
Algorithm:
1) determine A, B, C and D of F using $R_1$, $R_2$, $R_3$.
2) compute equation of line segment $\overline{P_1P_2}$.
3) determine the value of s using equation (2.6).
4) if s $\epsilon$ [0, 1]
      then return Q by computing equation of step (2)
      with the value of s obtained.
   else
      return null.
```

2.2.2 Determining the location of point of intersection

For a given plane F and a line segment g, if there is a point of intersection Q obtained between them then the location of Q i.e., whether it lies inside the $\triangle R_1R_2R_3$ or outside the triangle can be determined by

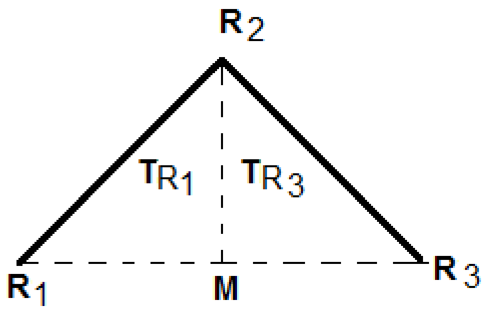


Figure 2.4: Location of point of intersection in $\triangle$R$_1$R$_2$R$_3$

representing the vector $\overrightarrow{R_1Q}$ as linear combination of vectors $\overrightarrow{R_2R_1}$ and $\overrightarrow{R_2R_3}$, which is given by

$$\overrightarrow{R_1Q} = \alpha(\overrightarrow{R_2R_1}) + \beta(\overrightarrow{R_2R_3}) \tag{2.7}$$

expanding the equation (2.7) for x, y and z coordinates gives three linear equations with two unknown variables, which upon solving gives the values of α and β.

If there exists a solution such that $0 \leq \alpha + \beta \leq 1$ and $0 \leq \alpha$, $\beta \leq 1$ then the point of intersection Q lies inside the $\triangle$R$_1$R$_2$R$_3$ and if $\alpha < \beta$ then Q lies in T$_{R_3}$ or if $\alpha \geq \beta$ then Q lies in T$_{R_1}$ otherwise Q lies outside the $\triangle$R$_1$R$_2$R$_3$. Here T$_{R_1}$ refers to $\triangle$R$_1$R$_2$M and T$_{R_3}$ refers to $\triangle$R$_3$R$_2$M as shown in Figure 2.4, where M is the Midpoint of $\overline{R_1R_3}$.

Algorithm 2 describes the steps for locating the point of intersection for given plane and line segment.

Algorithm 2 Determining the location of point of intersection

```
Input : 3 points $R_1$, $R_2$, $R_3$ which are used to define a plane
F and 2 points $P_1$, $P_2$ which define a line segment
Output: The point of intersection Q of the linesegment $\overline{P_1P_2}$
with the plane F if it is inside the triangle spanned by
$R_1$, $R_2$, $R_3$ and whether Q is in $T_{R_1}$ or in $T_{R_3}$
Restrictions: $R_1$, $R_2$, $R_3$ should not be collinear and $\overline{P_1P_2}$
should not be parallel to F
Algorithm:
1) determine A, B, C and D of F using $R_1$, $R_2$, $R_3$.
2) compute the point of intersection of F and $\overline{P_1P_2}$
   (refer Algorithm 1).
3) if Q exists, find the solution to
      $\overrightarrow{R_1Q}$ = $\alpha$ ($\overrightarrow{R_2R_1}$) + $\beta$ ($\overrightarrow{R_2R_3}$)
4) if 0 $\leq \alpha + \beta \leq$ 1 and 0 $\leq \alpha, \beta \leq$ 1
       if $\alpha < \beta$, return [Q, $T_{R_3}$]
       else if $\alpha \geq \beta$, return [Q, $T_{R_1}$]
5) else
     return null
```

Chapter 3

MANIPULATIONS

This thesis chapter focuses on ways to manipulate a random polygon. A generation of these random polygons was done by a research group that includes my thesis advisor. This work uses files which contain random polygons which were generated by this research group. For details on the generation process see [16, 17].

3.1 Reducer

3.1.1 Motivation and goals

Polygons generated in confinement are highly entangled and so a simplification routine is required to reduce the polygons to a more manageable form while preserving their topology (i.e., knot type). Preserving the knot type means that during the simplification process the segments of the polygon should not pass through each other.

The “Reducer” is a polygon simplification algorithm which includes geometrical manipulations and its goal is to eliminate vertices and to modify the polygon to a simplified form while preserving the knot type. More specifically, it either eliminates a vertex which is collinear with its two

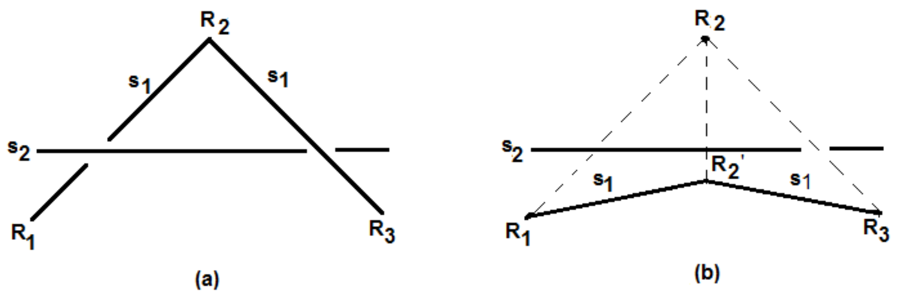


Figure 3.1: Displacement of vertex R_2

neighbors or moves the vertex repeatedly such that it becomes more collinear with its neighbors [11].

3.1.2 Approach

For a given triple of 3 consecutive non collinear vertices R_1, R_2 and R_3, R_2 of a polygon is moved towards the midpoint M of R_1R_3 [11]. The moved R_2 is referred to as R_2'. However, it must be ensured that moving R_2 to R_2' does not change the knot type of the polygon, which may happen if two segments s_1 and s_2 of the polygon as shown in Figure 3.1 pass through each other during the move.

The only two segments which are moved when R_2 is modified to R_2' are the segments $\overline{R_1R_2}$ and $\overline{R_2R_3}$ and are candidates for s_1. For s_2, only polygon segments which intersect the triangular plane spanned by R_1, R_2 and R_3 must be considered.

Thus the approach described in this thesis determines for each polygon segment g (excluding $\overline{R_1R_2}$, $\overline{R_2R_3}$ and the segments which are adjacent to R_1 and R_3) which intersects the triangular plane spanned by R_1, R_2 and R_3, the maximal possible displacement R'_{2g} along the segment $\overline{R_2M}$ (see Figure 3.2). $R_2{}'$ is then chosen as a random point on $\overline{R_2R_{2min}}$, where R_{2min} is the maximal displacement which satisfies $|$ R_2 - $R_{2min}| = min_g$ (R'_{2g}). Here M can be calculated as $\frac{1}{2}(R_1 + R_3)$ and the line segment from R_2 to M can be parameterized as $M + \lambda$ (R_2 - M). For this notation the smaller the value of λ the larger the displacement

Following is the algorithm which describes the procedure for moving one vertex of the polygon.

Algorithm 3 Displacement of a polygon vertex

```
Input: vertices $R_1$, $R_2$ and $R_3$ which are non collinear
consecutive vertices of the polygon.
Output: vertices $R_1$, $R'_2$ and $R_3$, where $R'_2$ is the
displacement of $R_2$.
Algorithm:
1) set Max$\alpha$ to 0.
2) determine the equation of plane F spanned by $R_1$, $R_2$ and $R_3$.
3) for each segment g in the polygon (except $\overline{R_1R_2}$ and $\overline{R_2R_3}$
   and adjacent vertices of $R_1$ and $R_3$)
      i)  determine the intersection point Q of the plane F
          and the line segment g (refer Algorithm 2).
      ii) if Q exists and Q is inside the triangle spanned
          by $R_1$, $R_2$ and $R_3$
              a) determine the max displacement $\alpha$ possible
                 for $R_1$, $R_2$, $R_3$ and Q (refer Algorithm 4)
              b) if $\alpha$ > Max$\alpha$
                        Max$\alpha$ = $\alpha$
4) pick a random r $\epsilon$ (Max$\alpha$, 1).
5) set $R_2{}'$ to M + r( $R_2$ - M ).
```

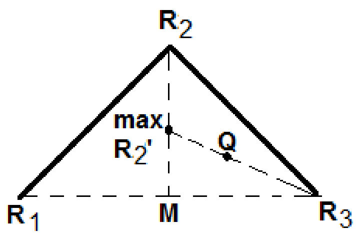


Figure 3.2: Maximum displacement of vertex R$_2$

3.1.2.1 Determination of maximal displacement of $\mathbf{R_2}$

The displacement of R$_2$ to R$_2{}'$ relative to a given segment g is maximal if Q (the intersection point of the triangle and g) is contained in the line segment $\overline{R_1R_2'}$ or $\overline{R_3R_2'}$ as shown in Figure 3.2. More precisely, if Q ϵ T$_{R_1}$ then max R$_2{}'$ is the intersection point of $\overline{R_1Q}$ and $\overline{R_2M}$. If Q ϵ T$_{R_3}$, then max R$_2{}'$ is the intersection point of $\overline{R_3Q}$ and $\overline{R_2M}$ (Algorithm 2 describes how to determine whether Q ϵ T$_{R_1}$ or Q ϵ T$_{R_3}$).

Following is the algorithm which determines the maximum displacement of R$_2$ to R$_2{}'$ relative to one segment g.

Algorithm 5 describes the implementation steps of Reducer. It first removes all vertices which are collinear with its neighbours. It repeatedly picks a random vertex R$_2$ which was not picked before and forms a triangle with neighbours of R$_2$ i.e., R$_1$, R$_3$, then uses Algortihm 3 to displace R$_2$ to R$_2'$. Final result is the reduced modified polygon.

Algorithm 4 Maximum displacement of R_2 to R'_2

```
Input: vertices $R_1$, $R_2$ and $R_3$ which are non collinear
consecutive vertices of the polygon and point Q which is the
point of intersection of a polygon segment g with the plane
spanned by $R_1$, $R_2$ and $R_3$.
Output: max displacement (expressed as $\alpha$) of $R_2$ to $R'_2$ which
is possible and Q remains inside the triangle $\triangle R_1R'_2R_3$.
Algorithm:
1) determine M, the midpoint of $\overline{R_1R_3}$.
2) if Q $\epsilon$ $\triangle$ $R_1R_2M$,
         solve $R_1$+ $s_q$( Q - $R_1$ ) = M + $\alpha$ ( $R_2$ - M ) for $s_q$, $\alpha$
   else
         solve $R_3$+ $s_q$( Q - $R_3$ ) = M + $\alpha$ ( $R_2$ - M ) for $s_q$, $\alpha$
3) return $\alpha$.
```

Algorithm 5 Reducer

```
Input : polygon with n vertices
Output: reduced modified polygon
Algorithm:
1) check collinearity of vertices i.e., for three consecutive
    vertices. Remove the middle vertex if they are collinear.

2) Repeat once for each vertex in the polygon
    i)  pick a random vertex $R_2$ which is not picked before
        and consider its adjacent vertices $R_1$, $R_3$ of the
        polygon.
    ii) displace the polygon vertex $R_2$(refer Algorithm 3).
```

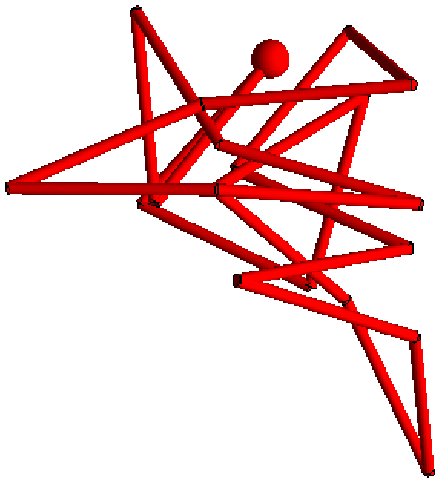


Figure 3.3: A 20 segment polygon before performing the Reducer. The ball shows the origin.

3.1.3 Results

Three different kinds of results of the Reducer are presented here.

1) The Reducer is applied to a polygon with 20 segments. After its execution the resultant is a simplied polygon which has 11 segments in it. Graphical representation [15] of this is shown in Figures 3.3 and 3.4.

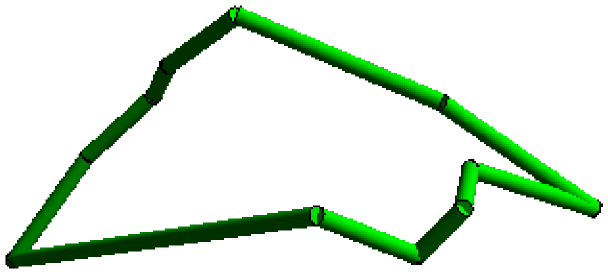


Figure 3.4: Modified polygon with 11 segments after performing the Reducer once. This polygon is unknotted and running the Reducer again several times will reduce it to a simple triangle.

Segments	Milliseconds
10	1.046784
20	1.863267
30	3.753503
50	9.440139
70	20.992357
90	41.089018
110	62.821073
140	90.427106
170	134.366771
190	186.66682

Table 3.1: Time taken for specified no.of segments to get reduced

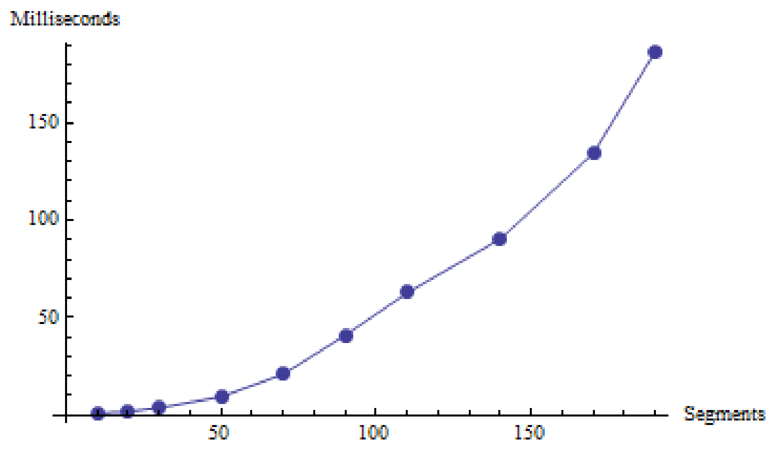


Figure 3.5: Time vs number of segments of Reducer

2) The second result shows the time taken for the Reducer to simplify an n segmented polygon. For this, we executed the Reducer on a hundred polygons of a particular length each 1000 times and the average the execution time is shown. The original polygons were generated in a radius of confinement equal to 3.

The data in the Table 3.1 depicts the number of segments versus time in milliseconds and its graphical representation [15] is shown in Figure 3.5.

Intial segments	Resultant segments
10	5.0
20	17.8
30	28.04
50	46.45
70	65.21
90	84.29
110	103.63
140	133.45
170	163.36
199	191.13

Table 3.2: Intial segments vs Resultant segments after performing Reducer

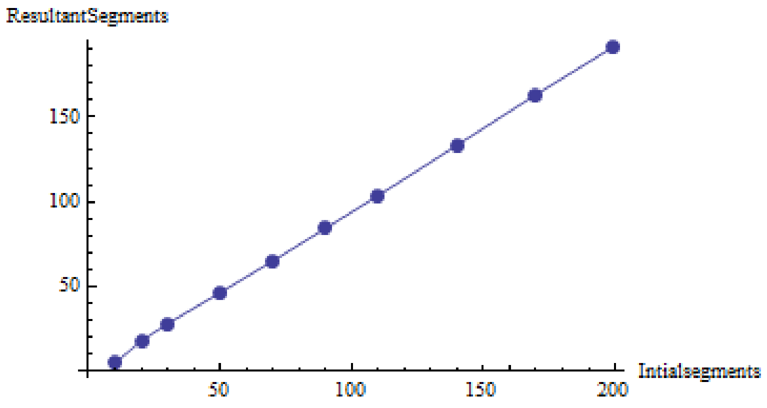


Figure 3.6: Intial segments vs resultant segments of Reducer.

3) The Reducer eliminates vertices of the polygons. The number of vertices after executing the Reducer is recorded. The data uses one hundred polygons and each polygon is reduced 50 times and the average number of resulting segments is computed. The data in the Table 3.2 depicts the number of segments versus resultant segments and its graphical representation [15] is shown in Figure 3.6.

3.2 Lengthener

3.2.1 Motivation and goals

Sometimes while trying to reduce a polygon to a more manageable form, the reducer cannot make any further changes. This is because, the polygon may get struck during the reduction process; as some of the segments of the polygon get much closer, which prevents the polygon from getting it reduced further.

The "Lengthener" is a polygon lengthening algorithm which includes geometrical manipulations and its goal is to relax and lengthen the polygon to an extent by adding vertices to the polygon . More specifically, segments are selected in random and vertices are added between them which enlarges the polygon. These vertices are added while preserving the knot type.

The approach in this section will make use of two different types of cylinders. Their definition is given here.

Original Cylinder

An original cylinder C_i as shown in Figure 3.7 is a cylinder having radius r, height h and its axis is the line segment from vertex v_i to v_{i+1} for $0 \leq i \leq n-2$ and C_{n-1} has its axis from v_0 to v_{n-1}. The radius r of the original cylinder is a fixed constant and the value of height h is equal to the distance between the vertices v_i and v_{i+1}

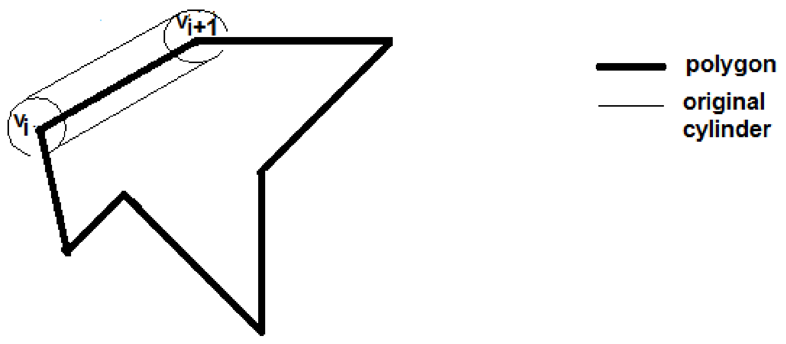


Figure 3.7: Original Cylinder C_i at random edge e_i

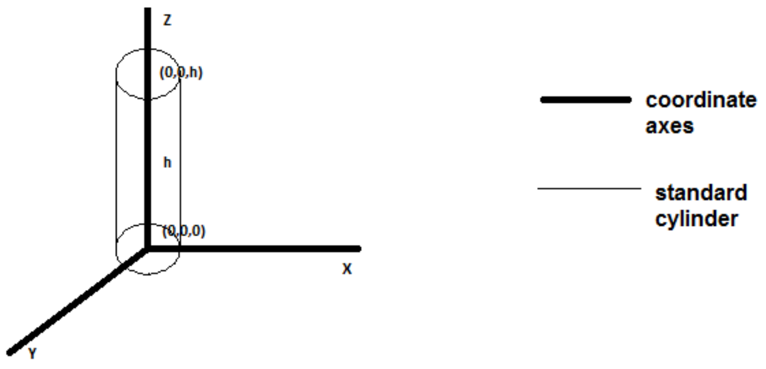


Figure 3.8: Standard Cylinder C_i^s

Standard Cylinder

Standard cylinder C_i^s for $0 \leq i \leq n-1$ has the same dimensions of the original cylinder C_i but the axis of the standard cylinder is the z axis such that one end of the axis is (0, 0, 0) and the other end is (0, 0, h) as shown in Figure 3.8, where h is the height of the original cylinder C_i.

3.2.2 Approach

One of the possibilities for lengthening the polygon is by adding vertices to it. Two consecutive vertices v_i and v_{i+1} of the polygon are selected and a random point is selected and added to the polygon; iff the knot type is not changed. The point to be added is randomly selected from a original cylinder around the segment $\overline{v_i v_{i+1}}$. A detailed explanation of this approach is as follows:

For two given vertices v_i and v_{i+1} which form an edge e_i, form an original cylinder C_i with e_i as its axis, and select a random point inside this cylinder. this requires two steps: First select a random point T inside the standard cylinder C_i^s, then use a transformation to compute corresponding point T' in C_i.

Algorithm 6 describes the generation of a random point T inside the standard cylinder C_i^s. In it, a random point on the base is selected in polar coordinates form (r', θ), where $0 \leq r' < r$ and also a random value h' on the height of C_i^s is selected, where $0 \leq h' < h$. Then the cartesian coordinates of T = (a, b, c) are determined by computing a = r'cos(θ), b = r'sin(θ) and c = h'.

Algorithm 6 Generating a random point in standard cylinder C_i^s

```
Input : radius r, height h
Output: random point T = (a, b, c)
Algorithm:
1) Pick a random angle $\theta$.
2) Pick a random value r', such that $0 \leq r' < r$
3) Pick a random value h', such that $0 \leq h' < h$
4) compute the coordinates of T = (a, b, c) using
   a =  r'cos($\theta$)
   b =  r'sin($\theta$)
   c =  h'
```

Transforming point T in C_i^s to a corresponding point T' in C_i involves calculating orthogonal vectors of C_i such that v_i, v_{i+1} is one of the base vectors and performing orthogonal transformation. For both are given as Algorithms 7 and 8.

Algorithm 7 Calculating Orthogonal vectors

```
Input: 2 points $v_i$, $v_{i+1}$ and a random point N
Output: 3 pair wise orthogonal vectors.
Restrictions: selected random point N should not be collinear
with $v_i$ and $v_{i+1}$
Algortihm:
calculating 3 orthogonal vectors
1) vector $p$ is nothing but the difference between the
given 2 points  i.e., $\overrightarrow{p}$ = $v_{i+1}$- $v_i$
2) now in order to determine the other two orthogonal
vectors $\overrightarrow{q}$ and $\overrightarrow{w}$, just pick the point N which is not
collinear with $v_i$ and $v_{i+1}$ then
$\overrightarrow{q}$ = ($v_i$ - N)$\times$($v_{i+1}$- N) (where $\times$ is the cross product)
is a vector orthogonal to $\overrightarrow{v_i v_{i+1}}$.
3) to determine the third vector $\overrightarrow{w}$ orthogonal to $\overrightarrow{p}$ and $\overrightarrow{q}$
compute $\overrightarrow{w}$ = $\overrightarrow{p}\times\overrightarrow{q}$ ( where $\times$ is the cross product)
4) now the unit vector's of $\overrightarrow{p}$, $\overrightarrow{q}$ and $\overrightarrow{w}$ i.e., $\hat{p}$, $\hat{q}$ and $\hat{w}$
are an orthogonal base.
```

Algorithm 8 Orthogonal transformation

```
Input:  2 points $v_i$, $v_{i+1}$ and a random point T = (a, b, c)
        inside $C_i^s$
Output: Transformation of T in $C_i^s$ to the corresponding
        point $T'$ in $C_i$
Algorithm:
1) compute orthogonal vectors $\overrightarrow{p}$, $\overrightarrow{q}$ and $\overrightarrow{w}$ for the given
   linesegment $\overline{v_i v_{i+1}}$ (refer Algorithm 7)
2) Transformation of T = (a, b, c) in $C_i^s$ to corresponding
   $T'$=($a'$, $b'$, $c'$) in $C_i$ can be represented as the
   linear combination of $\overrightarrow{p}$, $\overrightarrow{q}$, $\overrightarrow{w}$ and T = (a, b, c)
   and is given by
   $T' = \begin{bmatrix} a & b & c \end{bmatrix} \begin{bmatrix} p \\ q \\ w \end{bmatrix} + \begin{bmatrix} v_{ix} & v_{iy} & v_{iz} \end{bmatrix}$
```

In order to add a new vertex to the polygon, the first thing to do is to determine the equation of a plane spanned by v_i, T', v_{i+1} , where v_i and v_{i+1} are the end points of edge e_i and then check whether any line segments of the polygon (other than $\overline{v_iT'}$, $\overline{T'v_{i+1}}$ and also the adjacent vertices of v_i and v_{i+1}) cause an intersection with the $\triangle v_iT'v_{i+1}$ (refer Algorithm 2). If at least one intersection is found, then the random point T' cannot be added to the polygon as its addition might change the knot type of the polygon. If no intersection is found between triangle and the polygon segments, then the generated random point is added to the polygon. This process of selecting random edges potentially is repeated as often as desired.

Algorithm 9 describes the implementation steps for Lengthener. In it a random edge e_i is selected from among all the vertices of the polygon and an original cylinder C_i is formed with the selected edge e_i. A random point T, which is generated in the standard cylinder C_i^s is then transformed to corresponding T' in the original cylinder C_i which is formed with the edge e_i. For each line segment g of the polygon (excluding $\overline{v_iT'}$, $\overline{T'v_{i+1}}$ and segments which are adjacent to v_i, v_{i+1}), determine if there is any point of intersection inside the traingle inside $\triangle v_iT'v_{i+1}$ for the plane spanned by v_i, T', v_{i+1}. Upon calculation, even if there is at least one point of intersection found, then the generated random point T' inside the original cylinder C_i should not be added to the polygon because its addition changes the knot type as the edges of the polygon get crossed over each other. Otherwise, T' is to be added to the polygon. This process of selecting random edges potentially is repeated as often as desired.

Algorithm 9 Lengthener

```
Input: Polygon  with n vertices, m number of attempts,
r radius of cylinder
Output: Modified polygon
Algorithm:
Repeat m times
  1) Pick an random edge $e_i$ from among all the vertices
     in the polygon
  2) Generate a random point T inside $C_i^s$ (refer Algorithm 6)
  3) Transform T in $C_i^s$ to corresponding
     $T'$ in $C_i$ (refer Algorithm 8)
  4) for each line segment g of the polygon (excluding
       $\overline{v_iT'}$, $\overline{T'v_{i+1}}$ and segments which are adjacent to $v_i$, $v_{i+1}$)
       i)find the point of intersection Q inside $\triangle v_iT'v_{i+1}$ for
          the plane spanned by $v_i$, T', $v_{i+1}$(refer algorithm 2).
  5) In step 4, if alteast one point of intersection is found
        then do not add $T'$ to the polygon
     else
        add $T'$as a new vertex to the polygon in the cyclic
        order between $v_i$ and $v_{i+1}$
```

3.2.3 Results

Three different kinds of results of the Lengthener are presented here.

1) Lengthener is applied to a polygon having 6 segments in it . For this the value of m is 1 and r is 0.8. After its execution, the resultant is an enlarged polygon which has 12 segments in it. For this result graphical representation [15] of this is shown in Figures 3.9. In this Figure, the thin line represents the initial polygon i.e., polygon with 6 segments in it and the fat line represents the modified and enlarged polygon having 12 segments in it. The small sphere in this figure shows the starting vertex i.e., v_0 of both polygons .

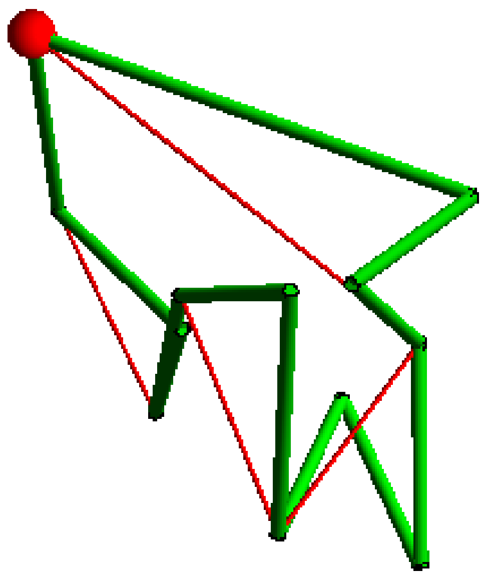


Figure 3.9: Given polygon modified to an enlarged polygon using Lengthener.

Segments	Milliseconds
10	0.217676
20	0.578496
30	0.818922
50	2.0126
70	3.74794
90	5.87855
110	8.55778
140	14.1267
170	19.8225
190	30.1593

Table 3.3: Time taken for specified no.of segments to lengthen

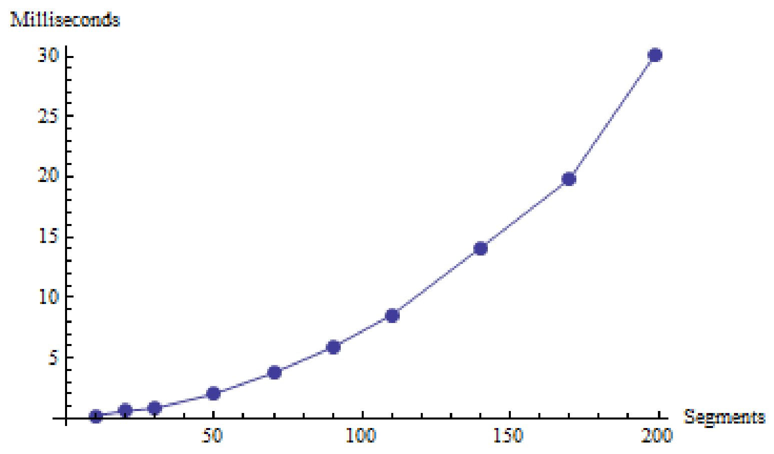


Figure 3.10: Time vs number of segments for Lengthener.

2) The second result shows the time taken for the Lengthener to enlarge an n segmented polygon. For this, we executed the Lengthener on one hundred polygons of a particular length each 1000 times and computed the average execution time. The radius of confinement of each original polygon is 3 and the radius of the cylinder C_i is 0.8 and the value of m is 1.

The data in the table 3.3 depicts the number of segments versus time in milliseconds and its graphical representation [15] is shown in Figure 3.10.

Intial segments	Resultant segments
10	20
20	40
30	60
50	98.33
70	139
90	178.97
110	220
140	280
170	338.62
199	380

Table 3.4: Intial segments vs Resultant segments

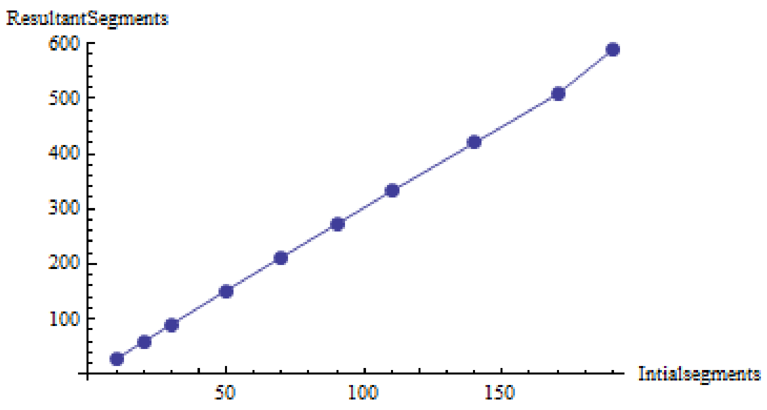


Figure 3.11: Number of segments versus resultant segments after performing Lengthener on an n segment polygon.

3) The Lengthener adds vertices to the polygon. For this, we executed the Lengthener on one hundred polygons of a particular length each 1000 times and computed the average of the number of segments. The radius of confinement of each polygon is 3 and the radius of the cylinder C_i is 0.8. The value of m is 1.

The data in the table 3.4 depicts the number of segments versus resultant segments and its graphical representation [15] is shown in Figure 3.11.

3.3 OutsideInLengthener

3.3.1 Motivation and goals

Analogous to the virus capsids confinement, in a sphere of a certain fixed radius polygons are generated [16, 17]. This sphere encloses the polygon in it which is packed compactly and the sphere when removed allows the polygon inside to expand by an algorithm such as lengthener. This expansion takes place in layers i.e., outermost layer of the polygon increases first, the underlying layer is extended next. Intuitively, the outer layer consists of the peripheral segments of the polygon, followed by the next layer and so on. This extension is implemented by adding vertices to the polygon.

The “OutsideInLengthener” is a polygon lengthening algorithm which includes geometrical manipulations and its goal is to relax and lengthen the polygon to an extent, by adding vertices to the polygon. More specifically, outer layer segments are selected first at random and vertices are added between them while preserving the knot type followed by an underlying layer and so on. All of this together relaxes and enlarges the polygon.

3.3.2 Approach

In order to lengthen and relax a given polygon, one possibility is to select an edge e_i randomly and insert a new vertex in the cyclic order between v_i and v_{i+1}. This new vertex is added only if there occurs no cross over between the edges of the polygon because of its addition to the polygon.

These edges are selected in the order of edges in the outer layer followed by the inner layer and so on. For dividing the given polygon into layers, the centroid C [1] of the polygon has to be found first and then the distances between the centroid and each edge is calculated. The polygon is then divided by grouping the edges according to their distance to the center of sphere. A detailed explanation is as follows:

3.3.2.1 Determining centroid of the polygon

The algorithm 10 describes the procedure for determining a centroid of the polygon [1]. In it the given polygon with n vertices is divided into n-2 triangles as shown in Figure 3.12 and the centroid R$_i$ and Area A$_i$ of each individual triangle is found, and we define a cummulative area T$_A$ as the summation of the areas of all individual triangles. Finally, the centroid of the polygon C is found as summation of area A$_i$ times the centroid R$_i$ of each individual triangle over T$_A$.

Algorithm 10 Determine the centroid of the polygon with n vertices

```
Input: Polygon with n number of vertices
Output: Centroid C of the polygon
Algorithm:
i)   Divide the polygon with n vertices into n-2 triangles T$_i$,
     the vertices of the triangle T$_i$ are {v$_0$, v$_i$, v$_{i+1}$}
     for 0 < i$\leq$n-2 (as shown in Figure 3.12).
ii)  Determine the Area A$_i$ and Centroid R$_i$ of each triangle
     T$_i$ for 0 < i$\leq$ n-2.
iii) Determine the T$_A$ = $\sum_{i=1}^{n-2}$ A$_i$.
iv)  Compute the centroid of the polygon to be C = $\frac{\sum_{i=1}^{n-2} A_i R_i}{T_A}$.
```

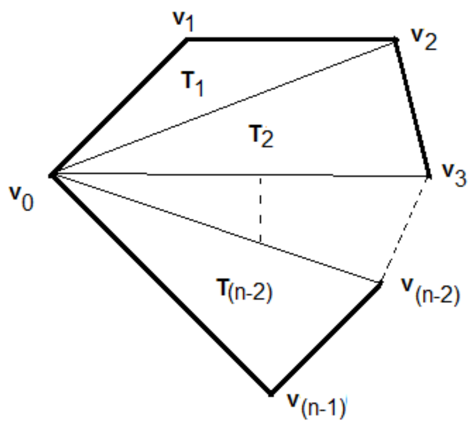


Figure 3.12: Division of polygon into n-2 triangles

Phase and Phase range

The segments of the polygon are divided into a constant number of disjoint groups based on the distance of the segments from the centroid. The groups are seperated by spheres centered at the centroid as shown in Figure 3.13. Each circular boundary is called a "phase", denoted as P and the region present between the two phases is called "Phase range", denoted by Ph. Categorization of edges e_i of the polygon into different phase P_k is done based upon the distance between (C, e_i)

3.3.2.2 Grouping edges into phases

Procedure for grouping the edges of the polygon into corresponding phases is described in the following steps

Step 1 : Determine the distance $d_i = |C - v_i|$ for $0 \leq i < n$.

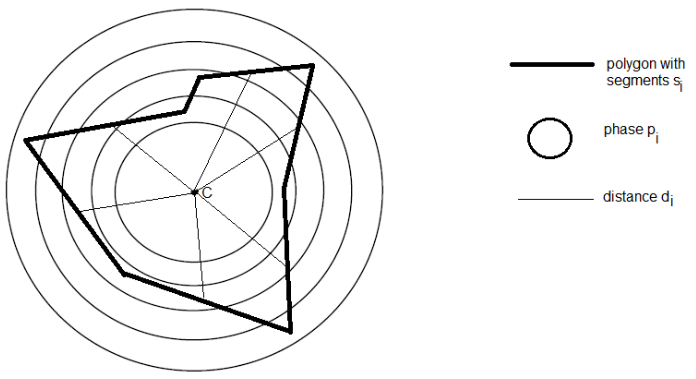


Figure 3.13: Division of a polygon into M phases

Step 2 : Calculate the segment distance s_i from the centroid as $s_i = \frac{(d_i+d_{i+1})}{2}$, for $0 \leq i \leq n\text{-}2$ and s_{n-1} is calculated as the mean of d_{n-1} and d_0

Step 3 : Determine the Ph for the polygon and which is given by Ph $= \frac{Max_s - Min_s}{M}$, where M is the number of phases, Max_s is the maximum value among the segment distances s_i and Min_s is the minimum value among the segment distances s_i.

Step 4 : If $Min_s + kPh \leq d_i < Min_s + (k+1)Ph$, for $0 \leq k < M$ then add segment i to phase P_k.

3.3.2.3 Adding a vertex

Segments are selected randomly from one phase at a time until all phases are processed in decreasing order. For each edge e_i, generate a random point T' in its original cylinder C_i. In order to add the new vertex T' to the polygon, first thing to do is to determine the equation of plane spanned by v_i, T', v_{i+1}; where v_i and v_{i+1} are the end points of edge e_i and then check whether any line segments of the polygon (other than $\overline{v_iT'}$, $\overline{T'v_{i+1}}$ and also the adjacent vertices of v_i and v_{i+1}) cause an intersection with the $\triangle v_iT'v_{i+1}$ (refer Algorithm 2). If at least one intersection is found, then the random point cannot be added to the polygon as its addition might change the knot type of the polygon. If no intersection is found between triangle and the polygon segments, then the generated random point T' is added to the polygon.

This process of selecting random edges and adding random points to the polygon has to be continued until the size of the vertices n of the given polygon becomes m (where m is an arbitrary number).

Algorithm 11 describes the implementation steps for OutsideInlengthener. In it, the centroid of the polygon is computed first. Then for each edge e_i of the polygon, the segment distance is calculated. The polygon is divided into M (where M is a constant number) number of phases as shown in Figure 3.12, each having a phase range Ph. Then the edges of the polygon are selected randomly one at a time from each phase where the order of phases is in decreasing order. For the selected random edge e_i original cylinder C_i is formed with it. A random point T, which is generated in the standard cylinder C_i^s is then transformed to corresponding T' in the original cylinder C_i which is formed with the edge e_i. For each line segment g of the polygon (excluding $\overline{v_iT'}$, $\overline{T'v_{i+1}}$ and segments which are adjacent to v_i, v_{i+1}), determine if there is any point of intersection inside the triangle inside $\triangle v_iT'v_{i+1}$ for the plane spanned by v_i, T', v_{i+1}. Upon calculation, even if there is at least one point of intersection found, then the generated random point T′ inside the original cylinder C_i should not be added to the polygon because its addition changes the knot type as the edges of the polygon gets crossed over each other. Otherwise, T′ has to be added to the polygon. This process has to be repeated m times (where m is an arbitrary number).

Algorithm 11 OutsideInLengthener

```
Input: Polygon with n vertices, a constant M for number
       of phases, m lower bound of vertices, r radius of
       cylinder
Output: Modified polygon
Algorithm:
while n < m
  1) Compute the centroid C of the polygon (ref Algorithm 10).
  2) For each edge $e_i$ of the polygon
        i) calculate the segment distance $s_i$, $0 \leq i \leq n-1$
            from the centroid C
  3) Divide the polygon edges into M phases based on their
     segment distances. The phase range of each phase is
      Ph = $\frac{Max_s - Min_s}{M}$.
  4) edges are selected randomly from one phase at a time
     and are processed in decreasing order of phases
  5) for each selected edge $e_i$
      i)Generate a random point T inside $C_i^s$ (ref Algorithm 6).
      ii)Transform T in $C_i^s$ to corresponding
         T′ in $C_i$ (refer Algorithm 8).
      iii)for each line segment g of the polygon (excluding
           $\overline{v_iT'}$, $\overline{T'v_{i+1}}$ and segments which are adjacent to
           $v_i$, $v_{i+1}$)
            a) find the point of intersection Q inside
               $\triangle v_i T' v_{i+1}$for the plane spanned by
                $v_i$, T′, $v_{i+1}$(refer algorithm 2).
      iv)In step iii, if at least one point of intersection
         Q is found
            then do not add T’ to the polygon
            else
              add T′ as a new vertex to the polygon in the
              cyclic order between $v_i$ and $v_{i+1}$
```

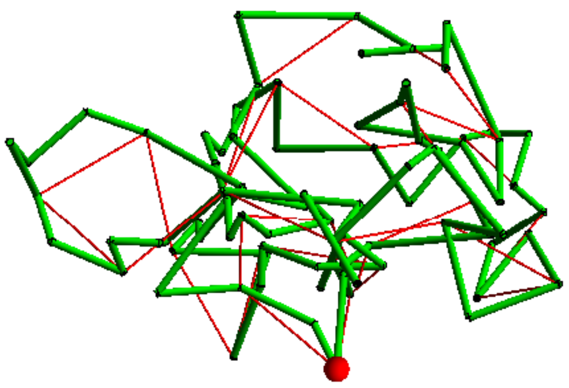


Figure 3.14: Given polygon modified to an enlarged polygon using OutsideIn-Lengthener

3.3.3 Results

Three different kinds of results of the OutsideInLengthener are presented here.

1) OutsideInLengthener is applied to a polygon having 30 segments in it. For this the value of m = 2n and r is 0.8. After its execution, the resultant is an enlarged polygon which has 84 segments in it. Graphical representation [15] of this is shown in Figures 3.14. In this Figure, the thin line represents the initial polygon i.e., polygon with 30 segments in it and the fat line represents the modified and enlarged polygon having 84 segments in it. A small sphere in this Figure shows the starting vertex i.e., v_0 of two polygons.

Segments	Milliseconds
10	0.322427
20	0.621988
30	1.03339
50	2.09295
70	3.51686
90	5.23074
110	7.50177
140	11.9996
170	16.1378
190	21.136

Table 3.5: Segments vs Milliseconds

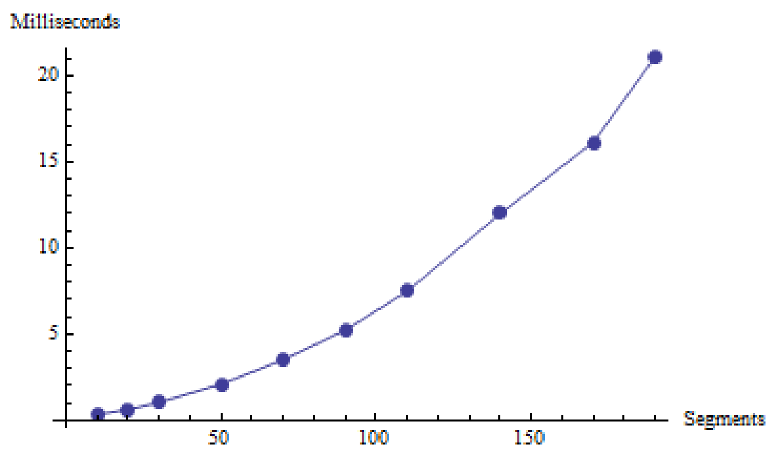


Figure 3.15: Time vs number of segments for OutsideInLengthener

2) The second result shows the time taken for the OutsideInLengthener to enlarge an n segmented polygon. For this, we executed the Lengthener on one hundred polygons of a particular length each 1000 times and computed the average execution time. The radius of confinement of each polygon is 3 and the radius of the cylinder C_i is 0.8 and the value of m = 2n.

The data in the Table 3.5 depicts the number of segments versus time in milliseconds and its graphical representation [15] is shown in Figure 3.15.

Intial segments	Resultant segments
10	28.0635
20	58.8725
30	90.1814
50	151.915
70	212.445
90	272.225
110	332.471
140	419.71
170	507.949
190	588.293

Table 3.6: Intial segments vs Resultant segments

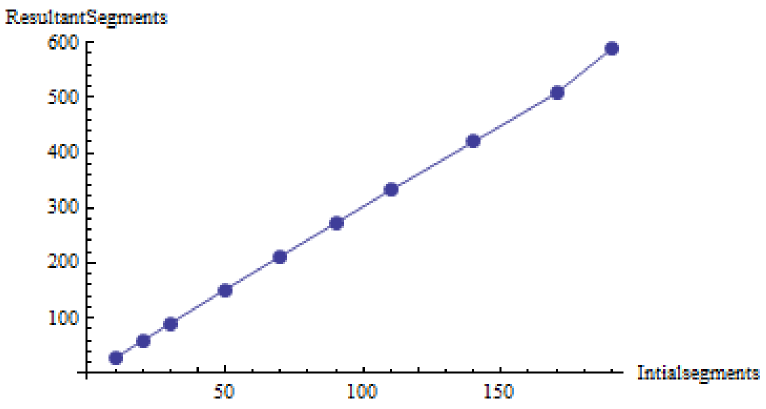


Figure 3.16: Number of segments versus resultant segments for OutsideIn-Lengthener

3) The OutsideInLengthener adds vertices to the polygon. For this, we executed the Lengthener on one hundred polygons of a particular length each 1000 times and computed the average no of segments. The radius of confinement of each polygon is 3 and the radius of the cylinder C_i is 0.8 and the value of m = 2n.

The data in the Table 3.6 depicts the number of segments versus resultant segments and its graphical representation [15] is shown in Figure 3.16

Chapter 4

FRAMEWORK

4.1 Framework

A software framework [18] is an abstract in which generic functionality is provided by the software and can be selectively changed by the user.

4.1.1 Motivation and goals

There is a need to investigate our polygons in a variety of ways and it is not clear which combination of manipulation steps might turn out to be the most useful. In order to accomplish this, for this thesis a framework is developed consisting of a list of user provided modules, which allows the user to manipulate the data by applying some or all the modules in a specified order. Additionally, the user can specify the number of repetitions for each module. The list of modules developed as a part of this thesis are Reducer, Lengthener, OutsideInLengthener. However, this framework is designed in such a way that other new modules may be added in the future with relative ease. Not all modules must manipulate polygons. Some may create a different representation of the polygon. For example from polygon to 2D embedding or vice versa, or the module which manipulate 2 dimensional representations or compute topological or geometrical feature of the represented data.

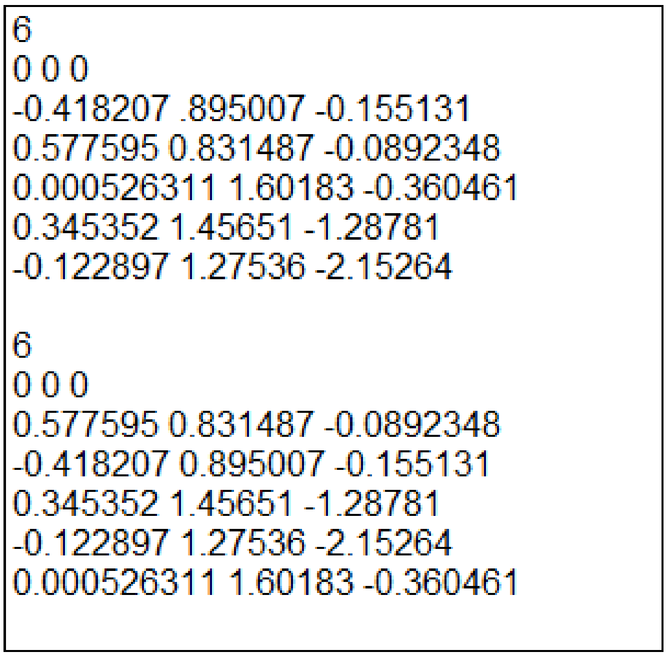
4.1.2 Approach

To achieve the stated goals, several tasks must be accomplished.

- For two or more modules to be executed in sequence the computed modifications of one module must be available to the next modules. This is accomplished by each module reading the information it needs from a file and writing the information generated to a file [14]. Thus, there are fixed file formats, a common data format which the polygon manipulator can read what another polygon manipulator writes is shown in Figure 4.1.

A polygon file may contain one or more polygons and each polygon with n vertices consists of n+2 lines in the file. The first line represents the number of segments the polygon has and the next n lines are the x, y, z coordinates of the n vertices and the n+2nd line is empty as shown in Figure 4.1.

- The user should be able to select which modules to apply, on which files it is to be applied to and in which order. The modules are executed in a flexible manner. This is accomplished by using a graphical user interface (GUI) [3]. This type of user interface is easy to



```
6
0 0 0
-0.418207 .895007 -0.155131
0.577595 0.831487 -0.0892348
0.000526311 1.60183 -0.360461
0.345352 1.45651 -1.28781
-0.122897 1.27536 -2.15264

6
0 0 0
0.577595 0.831487 -0.0892348
-0.418207 0.895007 -0.155131
0.345352 1.45651 -1.28781
-0.122897 1.27536 -2.15264
0.000526311 1.60183 -0.360461
```

Figure 4.1: Example of the file structure of two 6 step polygons

use, especially for a beginner and it is easy to explore. A GUI enables a user to view, control, and manipulate multiple items at once.

- The GUI should know about the different available modules. This is accomplished by forcing the developer of modules to put a folder containing their classes at a specific position within the overall directory structure of the framework [4]. For this thesis, the directory structure of the framework resembles a tree structure such that all manipulator modules are added to the same folder. This Folder structure is shown in Figure 4.2.

- The GUI must know the names of the important methods to use in each manipulator; otherwise it cannot execute them. This is

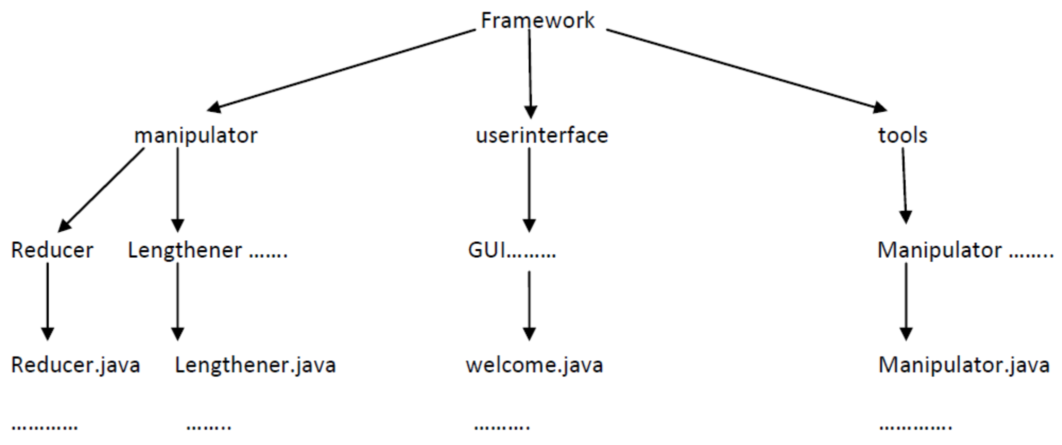


Figure 4.2: Directory folder structure of the Framework

accomplished by forcing the developers of modules to implement a specific interface [10]. In this thesis it is the Manipulator interface.

The flow of control of this framework is dictated by the Manipulator interface which defines a set of functions.

<<Manipulator>>

```
void readData(Scanner sc);
void manipulations();
void computations();
void writeModifiedDataToaFile(Scanner manipulations, Scanner computa-
tions);
String description();
```

a) void readData(Scanner sc)

This method is used for reading data from a text file. It is the responsibility of the person using the framework to make sure that a file contains the type of information needed by the manipulator module. This method has the Scanner object as its formal parameter and the developer is expected to store the data read from the file in some (private) class fields(s) suitable for the processing to be done by the module.

b) void manipulations()

If the class which implements the manipulator interface performs manipulations on the data from the input file, then the definition must be implemented; otherwise it should be left empty. This function takes no parameter. The developer is expected to use the field(s) where the data read from the file is stored.

c) void computations()

Computations do not manipulate the data but rather calculate the values of properties associated with the data, such as the mean path length or the radius of gyration. This function takes no parameter. The developer is expected to use the fields where the data read from the file is stored. The changes in the properties of the data can be better analyzed if computation methods are performed on the data before and after the execution of manipulation methods.

d) void writeModifiedDataToaFile(Scanner manipulations, Scanner computations)

After performing the necessary manipulations/computations or both, this method is helpful in writing the results to a file. This method has two formal parameters, the first is used for writing the modified data after performing manipulations on it into a text file and the other formal parameter is used for writing the results of the computations to a text file.

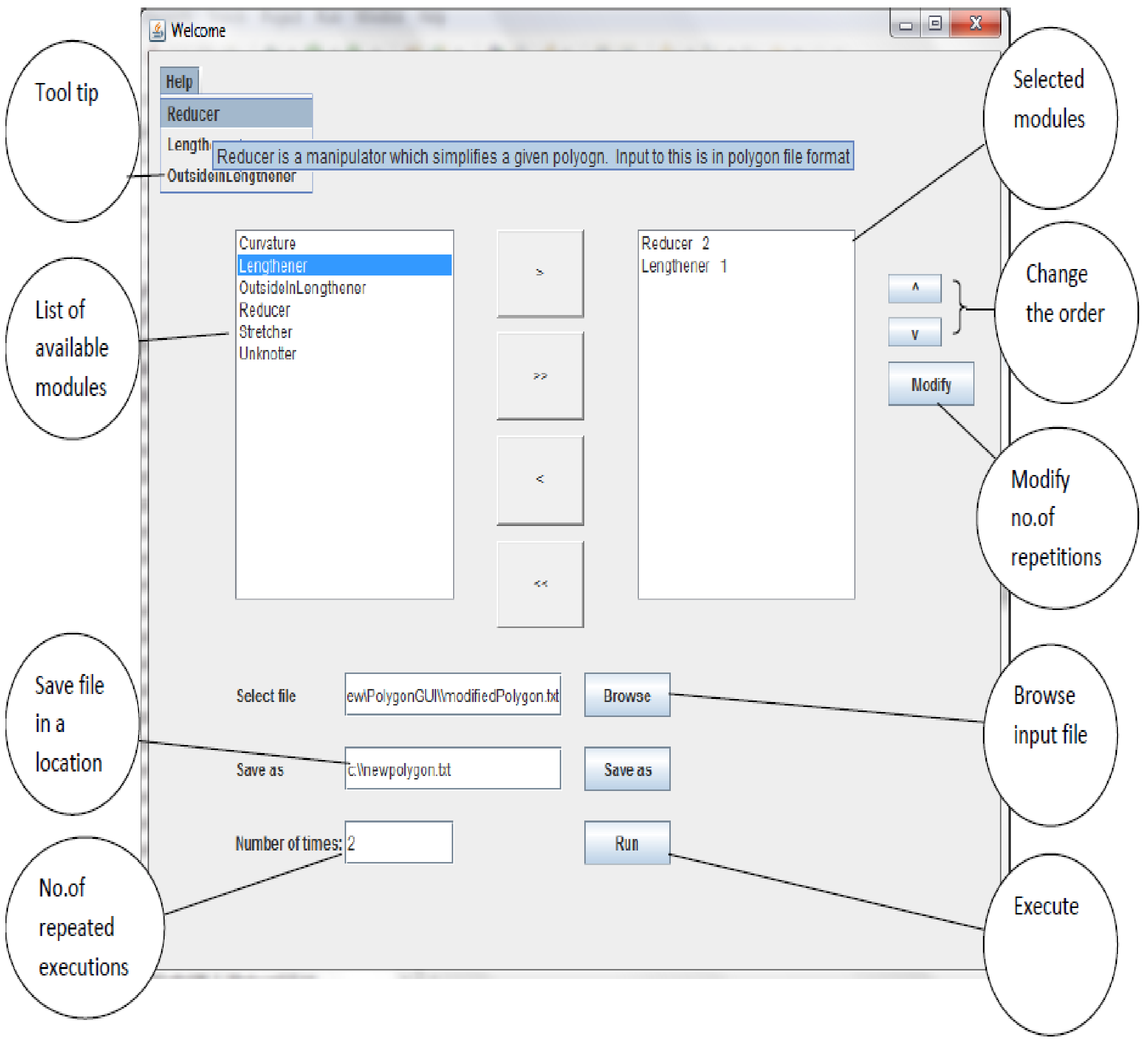
e) String description()

This method is used for providing some useful description to the user about the module. It includes whether the module is a manipulator/computation or both, what type of input the module needs and what type of output it needs. This method must return a string to the calling method which is used as input for tooltip in the GUI.

4.1.3 Graphical User Interface

4.1.3.1 Development of GUI

The GUI of the framework as shown in Figure 4.3 is developed using the Java programming language and the swing toolkit for java [3].



Reducer is a manipulator which simplifies a given polyogn. Input to this is in polygon file format

Select file: ew\PolygonGUI\\modifiedPolygon.txt

Save as: c:\\newpolygon.txt

Number of times: 2

Figure 4.3: Graphical user interface for this thesis

Look and feel of the GUI developed for this thesis

4.1.3.2 List of available modules

This list presents the user with a scrolling list of module names and is shown in Figure 4.4. The modules shown in this list are extracted from the manipulator folder in the framework directory structure [13]. The list is setup, so that the user can select one or more items at a time. Once the user clicks on the $>$ button, the selected items are copied to the selected module list. The $\gg$ button copies all the available items to the selected module list and for each module the user is prompted to enter the number of times the manipulate and the compute method is called per data set.

4.1.3.3 List of selected modules

This is the list of modules selected by the user as shown in Figure 4.5 from the list of available modules. GUI prompts the user to enter the number of repetitions while selecting a module. The user may change the order of the modules by selecting individual modules and using the up (Λ) and down ($\vee$) buttons until the modules are in the desired order. The *modify* button allows the user to change the number of repetitions for each module. Selected modules can be unselected by clicking the $<$ button and unselecting all of them can be done with the $\ll$ button.

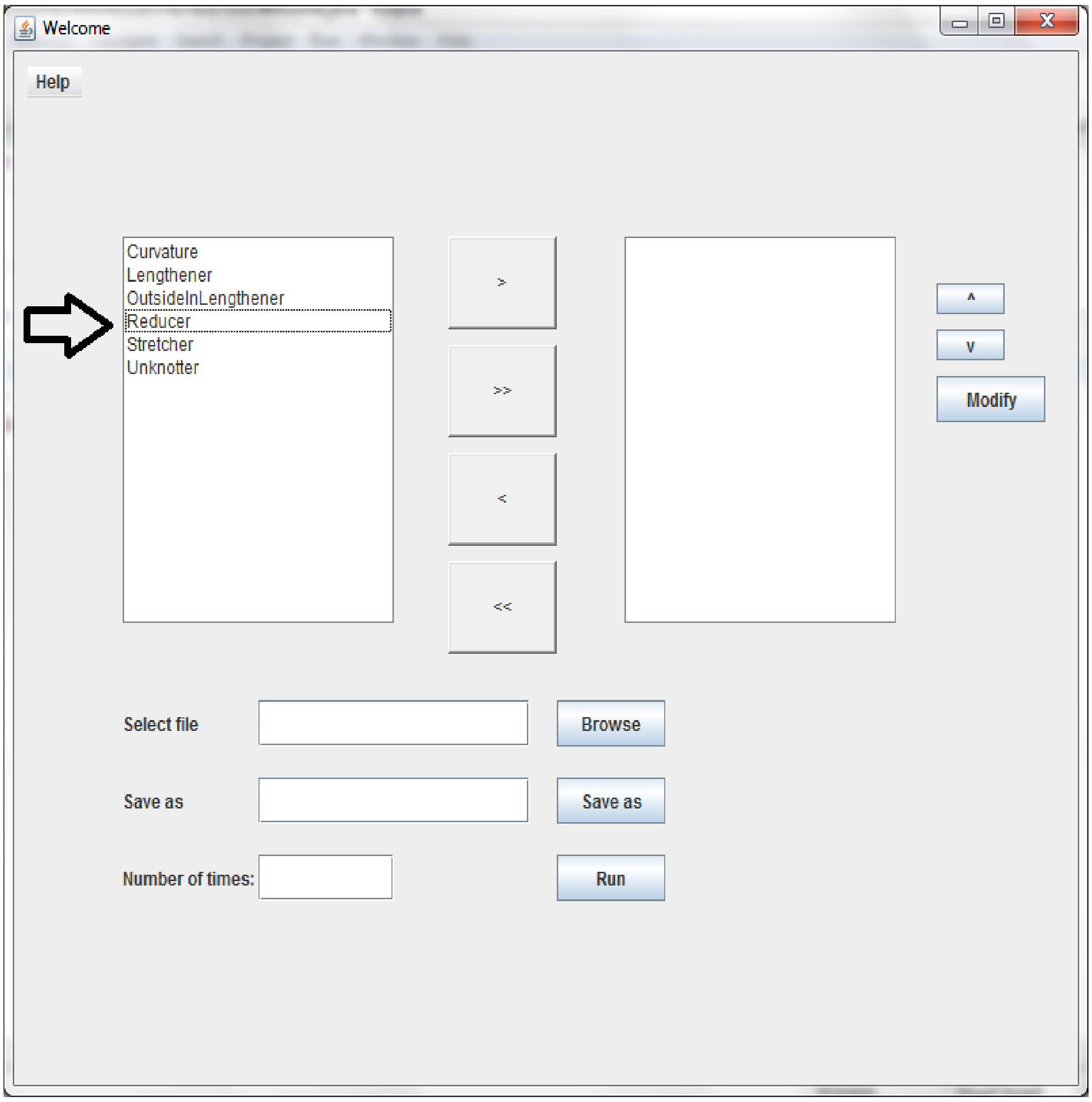
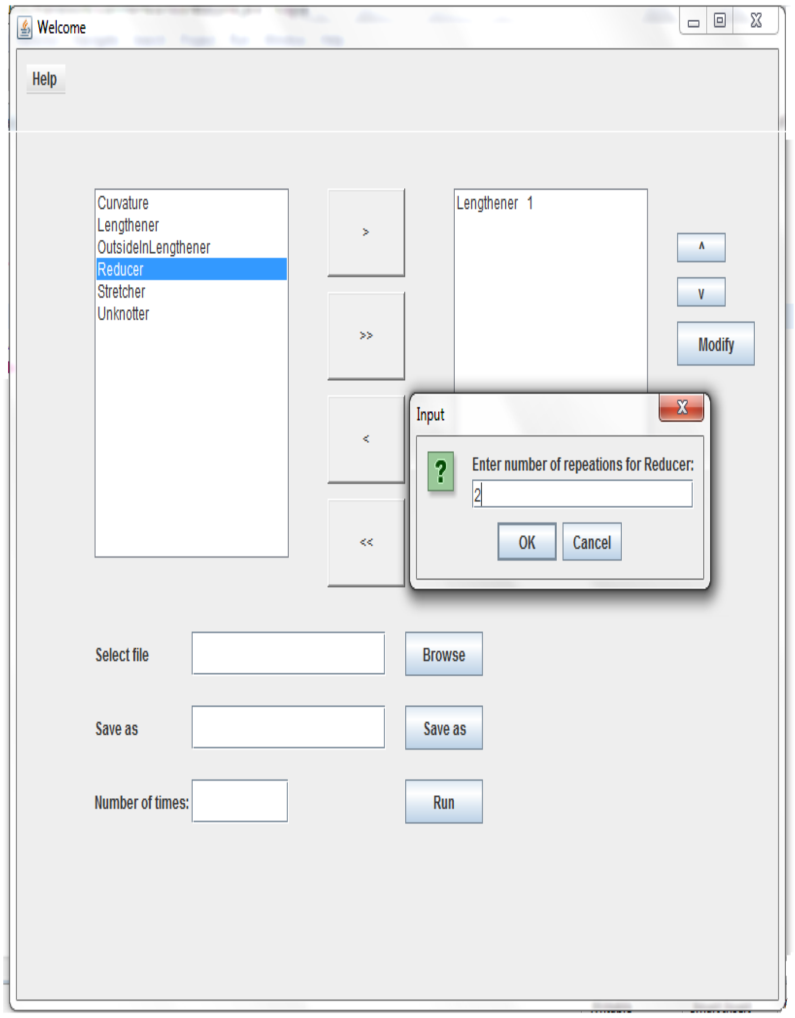


Figure 4.4: List of available modules



Lengthener 1

Input

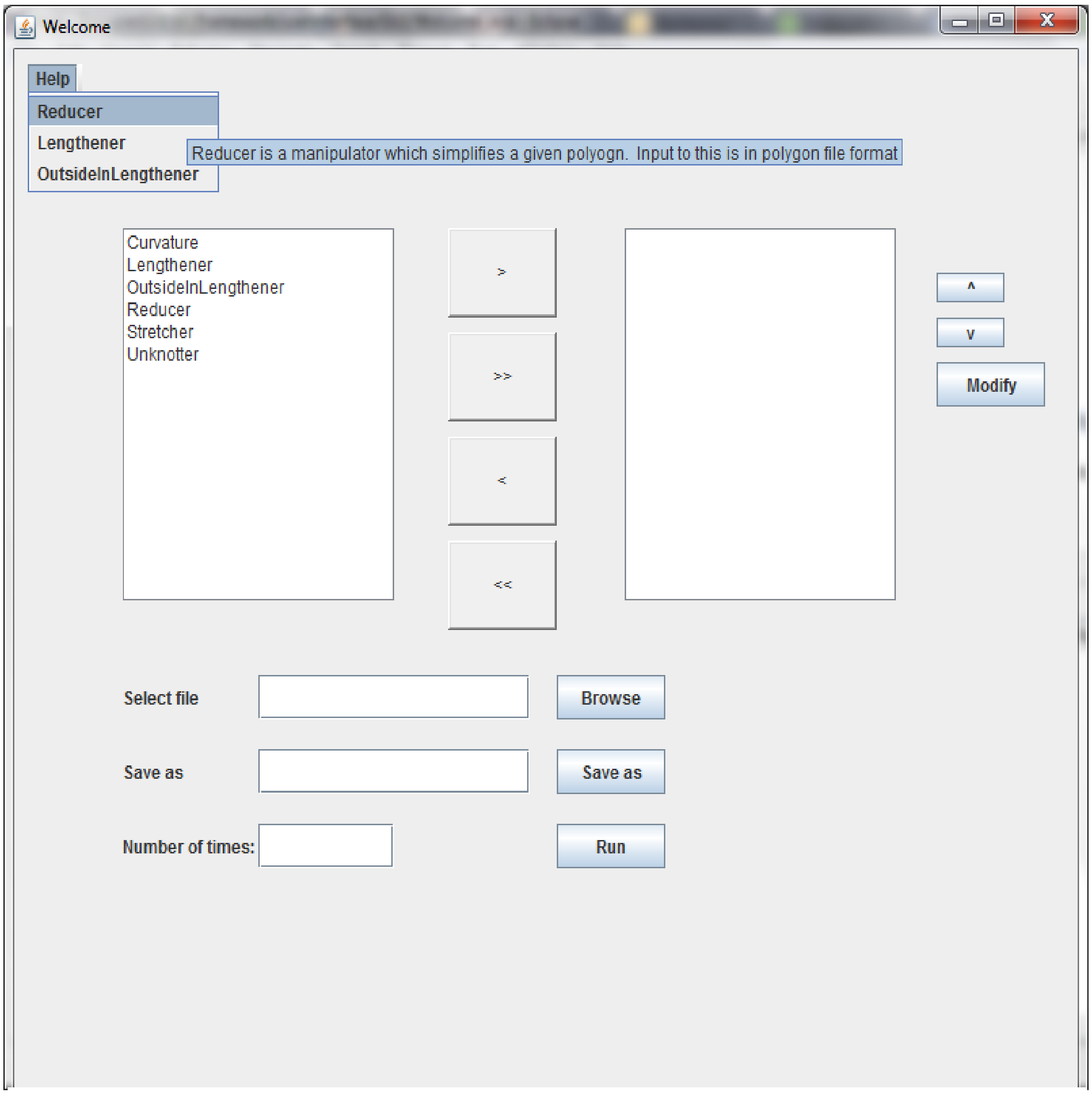
Enter number of repeations for Reducer: 2

Select file

Save as

Number of times:

Figure 4.5: Selected module list in GUI



Reducer is a manipulator which simplifies a given polyogn. Input to this is in polygon file format

Figure 4.6: Tooltip

4.1.3.4 Tool tips

The tooltip sometimes called info tip or hint, is a GUI element. It is used in conjunction with a cursor, usually a pointer. The user hover the pointer over an item without clicking it, and a small “hover box” appears with information about the item as shown in Figure 4.6. The tooltip provides the user with enough information to decide whether and how to include a module.

4.1.3.5 Browse a file

The user must specify the name of the file with the data to be used. For this the user has to click on the browse button which will show the file dialog box, then the user has to navigate to the file path- to select the file.

4.1.3.6 Save as

The save as button displays the “save as” dialog box, this gives the user a text box to assign a name to the file and a drop down list of bookmarks to select a directory to save it in.

4.1.3.7 Number of times

This text box is used for specifying the number of times the entire set of selected modules is repeated.

4.1.3.8 Run

Once the user is done with customizing the GUI and is ready to execute the selected modules, the only thing needed is to click the run button. This starts the execution, and prompts the user with a message box after the successful completion of the operation.

4.1.4 User actions

The user has to perform the following actions to run one or more modules in a given order. Here the “manipulation” refers to manipulation and/or computation.

- Select one or more modules from the list of available modules.
- Enter the number of iterations for each module when prompted.
- The list of selected modules is shown in the order of selection made by the user. Change the order of execution if needed, by using the up (Λ) and down ($\vee$) buttons until the modules are in the desired order.
- Modify the number of iterations for one or more modules if needed, by selecting the *modify* button and responding to the input dialog prompt.
- Browse the input file which contains the data on which the selected manipulations are to be performed.
- Name the output file in which the modified data is to be saved.
- After selecting the order in which the modules are to be executed; enter the number of repetitions of this order to be executed.
- Click the run button to launch the execution.
- Watch for the information about the successful completion of the execution.

Chapter 5

CONCLUSION AND FUTURE WORK

In this thesis, we focused on developing modules for performing manipulations on the polygon. Modules which were developed are Reducer, Lengthener, OutsideInLengthener. Reducer simplifies the polygon whereas Lengthener and OutsideInLengthener relaxes and enlarges the polygon to an extent by following different approaches. Performing these modules in any combination results in a modified polygon where the topology (i.e., knot type) is preserved.

A framework was also developed for this thesis, in order to investigate the data in a variety of ways. This framework includes the specified modules and allows the user to manipulate the data; by applying some or all modules in some specified order. Additionally, the user can specify the repetitions of each module.

Currently, other people (two undergraduates) are using this system and are working on the development of other modules like *embedder*, *curvature and torsion* which also perform manipulations or computations on the polygons.

Future work will focus on many different issues such as providing a condition loop for the number of repetitions of the execution instead of providing just a numerical number for it in the GUI. Also, another future work aspect is to make this framework available through a web link to the users who are willing to work in this area of research.

[1] J. M. Aarts. *Plane and Solid Geometry.* New York : Springer, 2008.

[2] Colin Adams. *The Knot book.* W.H Freeman and Company, 1994.

[3] James Elliott and Robert Eckstein. *JAVA SWING.* Oreilly and Associates Inc, 2003.

[4] Steven Holzner. *JAVA 2 Black Book.* CORIOLIS, 2001.

[5] http://paulbourke.net/geometry/disk (accessed 04.25.2012).

[6] http://softsurfer.com/Archive (accessed 04.25.2012).

[7] http://stackoverflow.com/questions/2678501/uniform-generation-of-3d-points-on-cylinder-cone (accessed 04.25.2012).

[8] http://www.wolfram.com/mathematica (accessed 04.25.2012).

[9] Bernard Kolman and David R.Hill. *Elementary Linear Algebra.* Pearson Education, 2004.

[10] Michael L. Laszlo. *Object-Oriented Programming Featuring Graphical Applications in Java.* Addison Wesley, 2001.

[11] Cristian Micheletti and Enzo Orlandini. Polymers with spatial or topological constraints: Theoretical and computational results, physics reports, volume 504, issue 1, july 2011, pages 1-73, issn 0370-1573.

[12] K. Richards and R. Calendar. Mode of dna packing within bacteriophage heads, journal of molecular biology, volume 78, issue 2, 5 august 1973, pages 255-256,in3-in5,257-259, issn 0022-2836.

[13] Herbert Schildt. *The Complete Reference for Java.* Mc GrawHill, 2009.

[14] Kathy Sierra and Bert Bates. *Head first java.* Oreilly and Associates Inc, Feb 2005.

[15] Bruce F. Torrence and Eve A. Torrence. *The students introduction to Mathematica.* Cambridge, 2009.

[16] Anthony Montemayor Yuanan Diao, Claus Ernst and Uta Ziegler. Generating equilateral random polygons in confinement i.

[17] Anthony Montemayor Yuanan Diao, Claus Ernst and Uta Ziegler. Generating equilateral random polygons in confinement ii.

[18] Heinz Züllighove. *Application-Oriented Software Development , San Francisco.* Elsevier Inc, 2005.