

Distribution of Unit Quaternions in S3 Space

by Sean Macken

FILE	14038_SEAN_MACKEN_DISTRIBUTION_OF_UNIT_QUATERNIONS_IN_S3_SPACE_78196_1315804577.DOCX (2.08M)		
TIME SUBMITTED	21-APR-2016 09:43AM	WORD COUNT	14423
SUBMISSION ID	56177080	CHARACTER COUNT	86302



Prifysgol Cymru
Y Drindod Dewi Sant
University of Wales
Trinity Saint David

Distribution of Unit Quaternions in S^3 Space

by

Sean Macken

Supervisor: Dr. Glenn L. Jenkins

Second Marker: Tim Bashford

Project submitted as part of the requirements for
the award of BSc (Hons) Computer Games
Development

April 2016

DECLARATION

I, Sean Macken declare that I am the sole author of this Project; that all references cited have been consulted; that I have conducted all work of which this is a record, and that the finished work lies within the prescribed word limits.

This work has not previously been accepted as part of any other degree submission.

Signed :

Date :

FORM OF CONSENT

I Sean Macken hereby consent that my Project, submitted in candidature for the B.Sc.(Hons) Computer Games Development degree, if successful, may be made available for inter-library loan or photocopying (subject to the law of copyright), and that the title and abstract may be made available to outside organisations.

Signed :

Date :

Abstract

Unit quaternions provide a way of representing rotations in a way that are free from singularities. This makes unit quaternions ideal when modelling anatomically correct joint models. The most recent work in this area attempted to use a rigid map that was designed to address the shortcomings from previous work which implemented a self-organising map. This paper proposes methods of distribution and measurement to aid in the creation of ideal initial training parameters for a rigid map.

Table of Contents

Abstract.....	1
List of Figures	4
List of Tables	6
1.0 Introduction	7
2.0 Literature Review	8
2.1 Artificial Neural Networks	8
2.1.1 Architectures of Artificial Neural Networks	9
2.2 Artificial Neural Learning	10
2.2.1 Correlation Learning	10
2.2.2 Competitive Learning	11
2.2.3 Feedback-based Weight Adaption	11
2.2.4 Supervised and Unsupervised Learning	12
2.3 Self-Organising Maps	12
2.4 Rigid Maps	13
2.5 Joint Models and Rotational Representation	14
2.5.2 Rotational Representation	14
2.5.2.1 Euler Angles	15
2.5.2.2 Exponential Maps	15
2.5.2.3 Quaternions	15
2.6 Previous Work	16
2.6.1 Subdivision to Create Geodesic Sphere	16
2.6.2 Polytopes	17
2.7 Distribution Methods	17
2.7.1 Uniformity and Normality	18
2.7.2 Uniform Rotations from Gaussians	18
2.7.3 Point on a Hyperellipsoid	19
2.8 Measuring Generated Points	20
2.8.1 Kurtosis	20
2.8.2 Skew	20
2.9 Summary	21
Chapter 3 – Research and Development Methodology	22
3.1 Research Methodology	22
3.2 The Experimental Hypothesis	22
3.4 Projects Inputs	22

3.5 Validity and Reliability.....	23
3.6 Conducting the Experiment	23
3.3 Development Methodology.....	23
3.4 The Hardware and Software in Use	24
Chapter 4 – Design and Implementation	25
4.1 User Requirements	25
4.2 System Requirements	25
4.3 Scenarios	26
4.3.1 Scenario 1 - Generating Patterns	26
4.3.2 Scenario 2 – Analysis of Patterns	27
4.3.3 Scenario 3 – Visualisation of Patterns.....	27
4.4 Use Case Documentation.....	28
4.4.1 Use case Diagrams	31
4.5 Design.....	33
4.5.1 Architectural	33
4.5.2 Mechanistic.....	37
4.5.3 Detail	47
4.6 Implementation	50
4.6.1 Testing.....	55
4.6.3 System Tests.....	58
4.6.4 Results.....	58
4.7 Results Discussion	74
Chapter 5 - Evaluation.....	77
Chapter 6 – Conclusion	78
Chapter 7 - Reflection	79
Chapter 8 - Bibliography	80

List of Figures

Figure 1: The four partial derivatives[9]	16
Figure 2: Tessellation of an equilateral triangle [14].	17
Figure 3: Box-Muller formula [20]	19
Figure 4: OSPHERE pseudo code [21].	19
Figure 5: Formula for calculating Kurtosis [22].	20
Figure 6: Use case diagram for generating patterns.....	31
Figure 7: Use case diagram for analysing patterns.	32
Figure 8: Use case diagram for visualisation of patterns.	32
Figure 9: Package diagram for the completed system.....	33
Figure 10: Class diagram for generating points.	34
Figure 11: Class diagram for analysis of points.	35
Figure 12: Class diagram for visualisation of points.	36
Figure 13: Sequence diagram for generation.	37
Figure 14: Ref for random method in generation sequence.	38
Figure 15: Random1D Ref.	38
Figure 16: Random2D ref.	39
Figure 17:Random3D ref.....	39
Figure 18: Random4D ref.	39
Figure 19: Box-Muller ref for generation sequence.	40
Figure 20: Box2D ref.	40
Figure 21: Box4D ref.	40
Figure 22: Fishman red for generation sequence.....	41
Figure 23: Fishman2D ref.	42
Figure 24:Fishman3D ref.....	42
Figure 25: Fishman4D ref.	42
Figure 26: Analysis sequence diagram.....	43
Figure 27: pack1D ref.	44
Figure 28:pack2D ref.....	44
Figure 29: pack3D ref.	45
Figure 30: pack4D ref.	45
Figure 31: Visualisation sequence diagram.	46
Figure 32: Activity diagram for Box-Muller 4D.	47
Figure 33: Activity diagram for Fishmans OSPHERE 4D.	48
Figure 34: Activity diagram for random 4D.....	49
Figure 35: State diagram showing states all the programs can enter.	50
Figure 36: Random distribution in one dimension code.....	51
Figure 37: Box-Muller code for one dimension.	51
Figure 38: Fishmans OSPHERE algorithm in two dimensions.	52
Figure 39: 3D visualisation of Fishmans OSPHERE.....	54
Figure 40: Mean and variance of non-normalised random distribution.	58
Figure 41: Skew and kurtosis of non-normalised random distribution.	59
Figure 42: Mean and variance from non-normalised OSPHERE.	60
Figure 43: Skew and kurtosis of non-normalised OSPHERE.....	61
Figure 44: Mean and variance of a modified non-normalised OSPHERE.....	62
Figure 45: Skew and kurtosis of a non-normalised modified OSPHERE.	63
Figure 46: Mean and variance of non-normalised Box-Muller.	64
Figure 47: Skew and kurtosis of non-normalised Box-Muller.....	65

Figure 48: Mean and variance of normalised random distribution.....	66
Figure 49: Skew and kurtosis of normalised random distribution.....	67
Figure 50: Mean and variance of normalised OSPHERE.	68
Figure 51: Skew and kurtosis of normalised OSPHERE.	69
Figure 52: Mean and variance of a modified normalised OSPHERE.	70
Figure 53: Skew and kurtosis for modified normalised OSPHERE.	71
Figure 54: Mean and variance of the Uniform Gaussian distribution.	72
Figure 55: Skew and kurtosis of uniform Gaussian.....	73

List of Tables

Table 1: Use case documentation for generating patterns.	28
Table 2: Use case documentation for analysis of patterns.	29
Table 3: Use case documentation for visualisation of patterns.	30
Table 4: Unit tests of vector class.	55
Table 5: Integration tests for list and vector.	56
Table 6: Unit tests for quaternion class.	57

1.0 Introduction

The aim of this project is to continue on in the research undertaken by Glenn L. Jenkins in which a Unit Quaternion based SOM is used in the aid of modelling anatomically correct joint restraints[1]. The work inside of this paper undertakes the testing and comparison of different types of training methods, most notably E-SOM and UQ-SOM.

The results from the experiment show that while the UQ-SOM can be used to help train the joint constraint, it does still not match up to the use of the E-SOM[1]. This is down to the way the unit quaternions are distributed through the network in which this projects main aim lies. Continuing on from this work this projects main aim is to find ways to uniformly or normally distribute unit quaternions throughout S3 space. To test the distribution of unit quaternions inside of S3 space different forms of metrics will be researched and looked at in order to test whether the distribution methods used have given a uniform or normally distributed sample size in multidimensional space.

The following chapters in this report will be a literature review covering previous major topics involved in this project, along with researched methods of distribution in multidimensional space with the objective of moving them into higher dimensions.

Covered in the literature review will also be the metrics researched and used to sample the distributed points to check their uniformity and normality. Following the literature review is a look into the research and development methodologies used for this project, which then leads into the design and implementation of the project itself.

2.0 Literature Review

The research inside of the literature review ranges from topics undertaken in previous work on this subject to topics in which the current aim of this project lies. The literature review will start by covering some of the most crucial background topics such as artificial neural networks and their training methods which leads into self-organising maps and rigid maps, which then moves onto how joints are modelled and constrained using different methods. Finally onto different forms of distributions that can be used inside of multidimensional space and how they can be measured.

2.1 Artificial Neural Networks

An artificial neural network often condensed down to ANNs is a computerised imitation of an actual neural network seen in animals[2]. These artificial neural networks are created for the purpose of helping a computer system replicate what an animal can do effortlessly such as recognise various objects and make sense of the large amount of visual data surrounding them, so it stands to reason that a computer system that is attempting a similar task would benefit greatly from understanding how it is done[2].

Some examples of this can be seen in the work by Rae and Ritter. In [3] it explains that while humans are easily capable of recognising where people are looking, it is much harder for a computer system to replicate this easy task [3]. Inside of their work they created a method that relied upon three layers of neural networks that were of the local linear map type which were used to enable the computer system to identify the orientation of a user by learning from examples [3]. These three layers each had their own specific role. The first neural network was used for colour segmentation, a second layer was used for localisation of the face, and the third layer was for the final recognition of the orientation of the head [3].

For a computer system to be labelled as an artificial neural network it is highly important that the system has a labelled directed graph structure in which nodes perform simple calculations [2]. Inside of an artificial neural network each node as stated above will be tasked with completing some simple calculation. From that calculation each connection to a node will convey a signal from one to another, often labelled by a number which is known as the connection strength or weight. This number indicates the extend in which the signal is amplified or diminished by a connection [2].

2.1.1 Architectures of Artificial Neural Networks

Inside of an artificial neural network it is not enough to have just a single node as it is insufficient for many of the practical problems that networks are designed to handle, because of this networks with large amounts of nodes are often used [2]. Due to the large amount of nodes the way in which they are connected determines how computations proceed inside of the network and just as biological neural networks have different structured nervous systems it is relevant to say the same in artificial neural networks and that it is incorrect to claim a single structured architecture can model every type of processing [2]. Below is an overview of the types of architecture available inside of artificial neural networks.

2.1.1.1 Fully Connected Networks.

The first type of architecture to look at is that of the fully connected network. Inside of this network every node is connected to each other, and these connections can either be excitatory, inhibitory or irrelevant[2]. Out of all the possible architectures possible for neural networks a fully connected network is the most general out of them all and every other architecture can be seen as a special case to it. However it is rarely used due to the large number of parameters inside of the network[2]. Due to this it is difficult to create a fast learning scheme inside of fully connected networks that are capable of generalising well[2].

2.1.1.2 Layered Networks

Alongside a fully connected network there are also those that are split into partitions or subsets known as layers[2]. A layered network for example works by having connections between nodes i , j and k however there cannot be a connection between these nodes if $i > j > k$. Inside of the layered network the first layer known as layer 0 or an input layer is where a single input arrives and is distributed to other nodes in the network through each node in layer 0, no other calculations are computed at this level and there are no intra-layer connections between nodes [2]. However unlike layer 0 (input layer) the final layer known as the output layer, connections with arbitrary weights can exist from any node in layer a to layer b for $b > a$. Inside of these layers intra-layer connections may also exist[2].

2.1.1.3 Acyclic Networks

An Acyclic Network is a subclass derived from the layered network. These two networks vary in that inside of an acyclic network there can be no intra-layer connections inside any of the layers of the network [2], this means that while a connection can be established between a and b while $a < b$, a connection cannot exist if $a = b$. Due to the simplicity of this networks architecture the computational process inside of acyclic networks are much simpler than those of other types of networks with exhaustive or intra-layer connections [2]. When a network is not acyclic they are referred to as recurrent networks.

2.1.1.4 FeedForward Networks

The feedforward architecture is a subclass of an acyclic network. These two networks differ by the feedforward network only allowing connections from nodes inside of layer a to nodes inside of layer $a + 1$ [2]. This type of architecture is often labelled by a sequence of numbers. These numbers are there to indicate the number of nodes inside of each layer. An example of this being a network labelled as 3-4-2-2. Where there are three nodes inside of the input layer, four nodes and two nodes on the two hidden layers and a final two nodes for the output layer. Generally feedforward networks will not have more than four layers, and are some of the most common neural network architectures currently in use [2].

2.1.1.5 Modular Networks

The final architecture is that of a modular based network. This network works by having several smaller modules with sparse interconnections between each module. The modularity of this network allows the neural network to solve smaller tasks independently using much smaller networks and then combine all of the modules back together in a logical manner. The modules are not limited in the way in which they can be connected and can be used in several different variations depending on the network itself [2].

2.2 Artificial Neural Learning

As artificial neural networks are modelled after real biological networks it is safe to say that just as humans and animals can learn so can an artificial neural network and just as there are multiple ways in which these artificial neural networks can be structured there are also multiple ways in which these networks are capable of learning about the data in which they hold.

2.2.1 Correlation Learning

Correlation learning often referred to as Hebbian learning is one of the oldest and most widely known principles for biological learning mechanisms which was described by Hebb

in 1949[2].

The process for correlation learning is as described.

When node a is close enough to excite another node b and repeatedly or persistently capable of exciting it, a growth process or metabolic change will take place in either one or both of the nodes, in doing so increasing the efficiency of a's capability of exciting b [2].

Inside of an artificial neural network this implies a gradual strength increase between connections of nodes that display similar outputs when they are presented with the same input. Due to these changes the strength between nodes represents the correlation between their outputs [2].

This weight modification rule is stated as:

$$\Delta w_{i,j} = c x_i x_j$$

Where c is a small constant which denotes the strength of connections and x_i and x_j are the activation levels of the specified nodes [2].

2.2.2 Competitive Learning

Another way in which neural networks can learn is through the use of competitive learning. This type of learning is often used inside of neural networks such as self-organising maps or rigid maps.

The competitive learning method works by nodes competing from input samples to become the winner, this takes place by self-excitation and mutual inhibition between nodes until a single winning node emerges from the input sample [2]. In doing so the winning node and its connections are then updated to increase the likelihood that the node wins in future input samples and competitions [2]. This competitive method and learning algorithm leads the network to develop nodes which specialise to be the winning node for sets of similar patterns [2].

2.2.3 Feedback-based Weight Adaption

Neural networks can also learn through a process known as feedback-based weight adaption. This form of learning works by increasing or decreasing the weights of the neural network based on a step size [2]. An example of this being if the neural network is within its first training step and increases the weights and at the end of the step the neural network sees a diminished performance the weights are then decreased by the step size to help try and improve the overall performance.

The step size for this training method is often set to a very small amount, this is to ensure that when the network makes a change to the weights that it does not stray too far from its currently evolved state. However if the step size is too small the neural network may require an excessively long training time [2].

2.2.4 Supervised and Unsupervised Learning

While an artificial neural network is learning in one of the ways stated previously, there are two ways in which these networks can go about learning from their input samples. The ways in which artificial neural networks can do this is through the use of supervised learning or unsupervised learning.

The first form is that of supervised learning. Supervised learning inside of an artificial neural network is commonly used when there is a known desired output from the sample training data [2]. Whereas inside of an unsupervised neural network the system is left alone without any interference to discover patterns or features that may be present inside of input data [2]. This leads onto the use of self-organising maps.

2.3 Self-Organising Maps

A self-organising map commonly referred to as a SOM is an unsupervised neural network in which combines a competitive learning algorithm with a topological structure of nodes, commonly used to simulate how biological neurons know how to organise themselves and replicate tasks such as pattern recognition without the use of external [2]. Due to the use of both a competitive learning algorithm and a topological structure, it is required that for the competitive learning algorithm to succeed nodes inside of the self-organising map must have inhibitory connections between all nodes while the topological structure demands excitatory connections to a small number of nodes inside of the network [2]. Inside of the network the topology is specified as a neighbourhood relation which exists between all nodes [2].

The self-organising map itself is a multiple layer network in which the first layer is the input layer while the second layer contains output nodes which contain many intra-connections set up to a predetermined topology, commonly a grid [2]. Each node inside of the output layer has a connection with every node within the input layer. These connections are given a strength value given by an n th dimensional vector w_j . These weights are initially given random values however during the learning process their values are updated and will commonly reflect a subset of input patterns [2].

When training the self-organising map the algorithm used updates a winner that is based upon an input sample, when the winner is located it is moved and so are the nodes inside of the networks topological distance. However over time during the training of the network the topological distance of neighbours is decreased, so as time progress less nodes in the network will be updated along with the winner, eventually leading to just the winning node moving [2]. As a result of this by the end of training the nodes inside of the network have now become ordered and the neighbouring nodes inside of the topology become associated with weighted vectors that are near each other in input space [2].

2.4 Rigid Maps

Following on from self-organising maps it was stated in [1] that while the unit quaternion self-organising map was capable of producing results to a reasonable degree of accuracy, that more research was required in the initial tuning of training parameters and the overall configuration of the unit quaternion self-organising map as a whole. This is where the use of rigid maps could be used to aid in the creation of better initial training parameters for an artificial neural network.

A rigid map is a variant of a self-organising map which may be used when the use of a self-organising map is not giving satisfactory results. To do this the rigid map modifies the internal and external topologies of the network and merges them into a single topology. Upon doing this, it is equivalent to setting the learning rate of a self-organising map to zero throughout the entire training process [4].

This change in topology affects the ways that nodes are now updated inside of the network, as now only the actual weight vectors of the nodes themselves are adapted during the training process, which leads to the term rigid map [4]. This change however also changes the way in which the winning node is now determined. The rigid map now requires the use of supervised learning, and the winning node is the node whose position inside of the network response space most closely matches that of the current training parameter [4].

Rigid maps however can only be used effectively if the response space of the network is known in advance, which is the case for pose estimations which leads into the next section about joint models and the different types of rotational representation. The best results are obtained from a rigid map when the known workspace has nodes that evenly cover the network response space [4].

2.5 Joint Models and Rotational Representation

As this project aims to research and develop methods of distribution which aid in the process of creating a more anatomically correct joint restraint it is important to first understand what a joint is and how they can be represented in space.

Inside of selected fields such as robotics [5], biomechanics and computer animation [6], hierarchical structures are frequently used in order to model bodies of humans, animals and even robots. These modelled bodies are made with sets of segments, with the segments being connected by joints. Joints are an essential feature as they allow as they allow relative degrees of motion between the connected segments, commonly referred to as a range of motion which can be characterised by a select number of parameters used to describe the motion in space and are constraint by joint limits[7].

An example of the simplest joint can be seen in [7] in which they talk about a revolute joint. The revolute joint model allows for a rotation about a single axis that is fixed in both of the connected segments and often within set angular limits. The revolute joint is one of the simplest as it is said to only have a single degree of freedom, commonly known as DOF, and due to its simplicity is one of the most used joints in current robotics [7]. This joint model works well for modelling simple joints of the human body such as the joints of hand or feet, however when it comes to modelling more complex joints such as the shoulders or hips, where there is more range of motion and more degrees of freedom the revolute joint will not work and more complex joint models are needed.

Deriving an accurate joint model from more complex joints is a difficult task as the joint will need a clear mathematical description of its allowed range of motion by giving it proper parameterisation. This is difficult due to the non-Euclidean nature of rotations, passing the proper parameters must be done carefully otherwise singularities may occur [7]. Secondly the joints range of motion should have a constraint placed on it by joint limits which are restricted to parameter space to a more realistic subset in which the joint can move [7]. This leads on to how to represent rotations inside of 3D space.

2.5.2 Rotational Representation

Inside of three dimensional space an object and its orientation can be parameterised in many ways as shown in [8] where they can be parameterised as Euler angles, quaternions and exponential maps to name a few.

As stated above selecting a way of representing rotations is difficult and can often lead to a situation in which a degrees of freedom are lost from a joint due to singularities occurring, creating gimbal lock of the joint [7].

2.5.2.1 Euler Angles

Euler angles are one way in which orientations of an object can be parameterised. Euler angles are one of the most established and popular forms of parameterisation. Generally when describing a rotation three mutually orthogonal coordinate axes are involved inside of fixed space. When a rotation happens around a single axis it is described as a roll [8]. Euler angles however ignore the interaction that happens between separate axis during a roll, this in turn creates gimbal lock of the joint as degrees of freedom have been lost [7].

2.5.2.2 Exponential Maps

An exponential map works by combining the axis and angle of a rotation into a single vector. This vectors direction represents the axis and the magnitude of rotation about that axis. However just as Euler angles, the exponential map will too suffer from singularities and produce gimbal lock in larger joints such as ball and socket in which there are three degrees of freedom that need to be described [9]. Exponential maps also have the drawback of no convenient method for combining multiple rotations. To fix this the exponential map must be converted into different formats, for example quaternions [8].

2.5.2.3 Quaternions

Although other forms of rotational representation exist this projects main focus for representing orientation in space is through the use of quaternions and their subset unit quaternions and how they can be uniformly or normally distributed to multidimensional space.

The quaternion is an extension of complex numbers and was invented by Hamilton. A definition of a quaternion is given by;

$$q = [s, v]$$

In which s is a scalar and v is a three dimensional vector [10]. Quaternions themselves form a group in which the underlying set is in four dimensional vector space known as R^4 , and their subset of unit-length quaternions commonly known as unit quaternions are set in S^3 , in which rotations can be carried out [9]. Unit quaternions are the main focus behind this project as rotations inside of S^3 space are free from singularities, therefore eliminating gimbal lock under controlled rotations. This is a result of how rotations using unit

quaternions are formulated. When calculating the rotation using unit quaternions a rotation matrix $R(q)$ is formed along with four partial derivatives, labelled as;

$$\partial R / \partial q_x, \partial R / \partial q_y, \partial R / \partial q_z, \text{ and } \partial R / \partial q_w$$

Figure 1: The four partial derivatives[9]

These four derivatives all exist within S3 space and are all linearly independent all over S3, meaning that when used for rotations quaternions are capable of producing gimbal lock free systems as they will not produce singularities. However for this to work it relies on quaternions staying within S3 space, meaning the quaternions must maintain their unit length throughout the process. This is due to how quaternions have a total of four direction in which they can change but only have three rotational degrees of freedom which can lead to quaternions being pushed off the unit quaternion sphere and outside of S3 space [9].

2.6 Previous Work

2.6.1 Subdivision to Create Geodesic Sphere

Earlier work that showed a way of uniformly distributing points was that of subdividing a platonic solid to create a geodesic sphere whose midpoints should uniformly cover the sphere they are on. While other forms of tessellation and subdivision were looked at such as the spiral tessellation technique used in [11] or the tessellation of a unit sphere with a convex triangular mesh in [12] the creation of a geodesic sphere was chosen due to the uniformity of its faces and potential midpoints.

A platonic solid occurs when a polyhedrons faces are all equal and regular, while the vertices are all surround alike in which five shapes currently exist. These five shapes are the tetrahedron, dipyramid, octahedron, icosahedron and finally a dodecahedron [13]. The chosen platonic solid to be used in creating the geodesic sphere was the icosahedron.

The icosahedron was used in this method as it holds the three required features needed in creating the geodesic sphere along with being the only platonic solid to closely resemble a sphere itself. These features are that all of the triangles forming the shape are equilateral, all are the same size and that all angles that are made are equal angles with one another [14].

The geodesic sphere itself is formed through the process of subdividing the icosahedron multiple times. Due to the nature of the icosahedron and its equilateral faces, subdividing

of the faces is easy and will always produce an outcome with a perfect square as shown below.

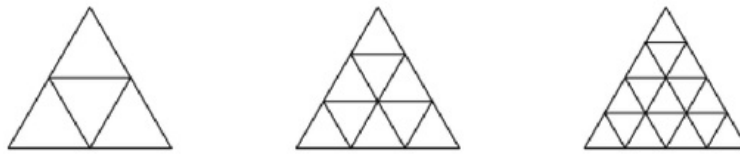


Figure 2: Tessellation of an equilateral triangle [14].

However simply subdividing the icosahedron multiple times will not generate the geodesic sphere. Although the current vertices of the original icosahedron will lie on the surface on the sphere, any new points that are created will be created inside of the planes on which those triangles exist. This will have the effect of the newly created vertices being inside of the sphere, the solution to this problem however is to push those points from inside of the sphere onto the surface from the centre of the sphere.

2.6.2 Polytopes

The latest look into this subject was undertaken by George in [15] in which he tried to improve the results given by the rigid map by introducing polytopes. Georges work took the proposed topology by Winkler in his paper [4]. This topology was formed using seven hundred and twenty six nodes. Six hundred of these nodes formed the vertices of a dodecaplex, commonly known as a 120-cell while the other one hundred and twenty nodes created the vertices of a hexacosichoron, known as a 600-cell [15]. Using this topology an even distribution of points about the unit 3-sphere was created.

Upon conducting tests George concluded in his work that the rigid map was capable of constraining bounding regions for a joints swing with low error if certain network settings were used. Although it was stated that these setting were difficult and time consuming to find and often changed the behaviour of the network as a whole [15]. The tests also showed that the network when trained was not capable of representing twist rotations without poor response time and a high error rate, in which he concluded the rigid map alone was not enough to undertake this task [15].

2.7 Distribution Methods

This next section covers researching different types of distribution that will work within multidimensional space with the aim of implementing them into four dimensions and producing data that is either uniformly or normally distributed.

2.7.1 Uniformity and Normality

Uniformity and normality are both descriptive terms in which describe the probability of an occurrence of points. The normal distribution is one of the most important continuous probability distributions used in engineering and science and can be used to describe many of the phenomena that occur in nature [16]. The properties of a normal distribution are that the mode of the data will be on a horizontal axis will always equal μ , which in the case of a normal distribution is zero. The curve will also be symmetrical on the vertical axis around the mean, and the curve approaches the horizontal axes asymptotically as you move either direction from the mean and the total area under the graph adds up to the probability of one [16].

Data is said to be uniformly distributed when a random variable X is restricted to an interval of finite size such as $[a,b]$ and has the function $f(x)$ in which has a constant density over the interval [17]. Unlike the normal distribution that peaks around the mode of the data, uniform distribution has no mode. If visualised in a graph the normal distribution will take the form of a bell shaped curve, whereas the uniform distribution will produce a flat line within its interval limits.

2.7.2 Uniform Rotations from Gaussians

One way to generate uniform points has been shown in [18] in which random quaternions are created through the process of being assigned four independent Gaussian distributed points. These Gaussian points are generated with a mean of zero and a common standard deviation, which in the case of this project will be one. When the four independent variables are assigned to the quaternion the quaternion itself will be will be a four dimensional Gaussian distributed point which can then be normalised to create a uniformly distributed unit quaternion [18].

A way of creating pairs of independent variables is shown in [19] in which a polar method is used however this project will use a variant to this method known as Box-Muller. The Box-Muller method just like the polar method is capable of producing pairs of independent variables that are Gaussian distributed and can be seen in [20].

The Box-Muller method works by taking two independent uniform random variables that are within the uniform range of $[0, 1)$, these points commonly labelled as $U1$ and $U2$. To create the two independent pairs from these numbers a mathematical operation shown below is used on the numbers to generate the points $Z0$ and $Z1$.

$$Z_0 = \sqrt{-2 \ln U_1} \cos(2\pi U_2)$$

$$Z_1 = \sqrt{-2 \ln U_1} \sin(2\pi U_2)$$

Figure 3: Box-Muller formula [20].

Repeating this method of distribution twice will produce four independent variables of which can then be assigned to a quaternion. The final approach then as stated earlier is to normalise the quaternion creating a uniformly distributed quaternion.

2.7.3 Point on a Hyperellipsoid

Another distribution method looked at was one mentioned by Fishman in [21]. The method of distribution Fishman uses to distribute points uniformly over the hypersphere is that of the OSPHERE algorithm.

The OSPHERE algorithm works by taking two inputs. The first input being the number of dimensions, labelled as n , the second input being the radius of the unit sphere r , in this case one. Once the method has two parameters passed to it can begin creating vectors of n -dimensions. Figure 4 shows the pseudo code used in generating these vectors.

```

 $i \leftarrow 1; W \leftarrow 0.$ 
Until  $i > n$ :
  Randomly generate  $X_i$  from  $\mathcal{N}(0, 1)$ .
   $W \leftarrow W + X_i^2.$ 
   $i \leftarrow i + 1.$ 
 $W \leftarrow \sqrt{W}.$ 
 $i \leftarrow 1.$ 
Until  $i > n$ :  $Z_i \leftarrow rX_i/W$ ;  $i \leftarrow i + 1.$ 
Return  $\mathbf{Z}.$ 

```

Figure 4: OSPHERE pseudo code [21].

The OSPHERE algorithm works generating an X value for the number of dimensions given to the method. As an example using quaternions, four X values will be generated within a normal distribution of mean 0 and standard deviation of 1, with W becoming the sum of each X value squared. Once the generation of X values has finished, W will become the root of itself and all four values of the Z vector can be assigned by multiplying the corresponding X values by the radius and then dividing them by W .

The purpose of this method is that given a number of dimensions over one, as it is impossible to have a sphere or circle with a single dimension, that it can generate a given

number points by repeating the process and these points will be uniformly distributed in their given dimensions.

2.8 Measuring Generated Points

To sample and measure the points generated by the different forms of distribution two methods of analysis have been chosen and those are the skew and kurtosis values of the generated points

2.8.1 Kurtosis

Kurtosis is a method in which characteristics of distributed data can be captured. The main characteristics captured by kurtosis are the central tendency, variability and shape of the distributed points [22].

Kurtosis itself can be defined as shown in figure 5;

$$\beta_2 = \frac{E(X - \mu)^4}{(E(X - \mu)^2)^2} = \frac{\mu_4}{\sigma^4},$$

Figure 5: Formula for calculating Kurtosis [22].

Kurtosis has well known distributions when calculated, the two known distributions this project aims to achieve is the value of either -1.2 or 0. Having a value of -1.2 for kurtosis will indicate that the sampled data is uniformly distributed throughout its given space, whereas a value of 0 will indicate that the sampled data has been normally distributed [22].

2.8.2 Skew

Measuring the skew of sampled data allows for the analysis of symmetry in the chosen sample size. The skew of data can commonly be visualised through the use of histograms and graphs as seen in figure 2 in [23]. As this projects aims to create normally or uniformly distributed points, if the data was to be visualised as show within [23] for a normal distribution the bell-shaped curve would be expected due to as numbers increase or decrease away from the mean the probability of them occurring lowers. Whereas if the data was uniform the expected graph would be flat as it would have no mode and all numbers share the same probability of occurring.

While calculating the skew of sampled data it is possible that the calculation returns either a negative value, positive value or zero. If the skew of the sampled data is zero then the distribution is symmetric, however if the skew returns as a negative value it means more

data is concentrated on the right side of the mean of the data set, where as if the skew is positive more data is concentrated to the left of the mean of the data set [23].

2.9 Summary

Many topics have been covered in the literature review, this project will mainly focus on the topics discussing the uniformity and normality of data as well as the methods of distribution which claim to produce the desired result when used in four dimensions. The metrics of measurement are also a main focus point, as analysing the skew and kurtosis of generated patterns will also allow for the distinction of how these generated patterns are distributed and whether or not they meet the required values for uniformity and normality.

Chapter 3 – Research and Development Methodology

3.1 Research Methodology

The chosen research methodology for this project was that of a literature review. A literature review was chosen for this project over other methodologies due to this project being experimental and aiming to test a hypothesis. As this project tests a hypothesis a literature review was the ideal choice as a research methodology as it allowed for the critical review of previous work undertaken in this subject. Reviewing previous work gave detailed explanations into how and what were tested during earlier implementations of this project and the results that came out of the testing.

Having detailed descriptions and visual analysis of earlier tests and their results allowed for a more refined development process which aimed to replicate earlier tests and improve upon their results.

3.2 The Experimental Hypothesis

As this project is an experimental based project it is based around a hypothesis in which needs to be tested and evaluated to see whether or not the proposed hypothesis is correct. This project aims to research and develop distribution methods in multidimensional space moving up into four dimensions. The hypothesis is that when these distribution methods are moved up into four dimensions they will be capable of producing patterns of data which are either uniformly or normally distributed throughout four dimensions.

3.4 Projects Inputs

Due to the automatic nature of the project in which data patterns are generated in multidimensional space through different distribution methods and then analysed to determine characteristics, the inputs for the project were rather small. For the generation program, inputs ranged from three integers in which the first controlled the method of distribution which a valid range of one to three. The second integer controlled the amount of points to generate with a range of one to a few thousand and the final integer determining the dimension selection with valid ranges of one to four.

The analysis and visualisation programs to suffer from limited input due to an automatic nature, both analysis and visualisation only take two launch parameters. The first parameter being an integer labelling the method of distribution to use and the second being another integer used to determine the dimension used.

3.5 Validity and Reliability

To ensure this project was valid, when conducting tests each distribution method was tested a total of ten times, each time measuring the mean, variance, skew and kurtosis and at the end calculating a mean for each statistic. Reliability was ensured by reseeding the number generators inside of the generation program every time it was run. This ensures that when running any generation method the number generators would be randomised based upon the system clock in which it is not possible to repeat results.

3.6 Conducting the Experiment

The experiment was conducted by taking the results gathered from the repeated tests. Each distribution method was then placed into two separate graphs. These graphs displayed the mean and variance of a distribution method on one graph from one to four dimensions while the second displayed the skew and kurtosis of distribution methods in one to four dimensions.

3.3 Development Methodology

The development methodology chosen for this project was a modified version of Rapid Object-Orientated Process for Embedded Systems, commonly condensed ROPES. The ROPES development cycle works based upon an iterative lifecycle which uses a spiral approach to development compared to the usual waterfall method seen in many other development methodologies which can suffer from incompleteness. Incompleteness occurs when the artefacts of any phases are not capable of being completed until they are elaborated on in subsequent phases and their problems have been identified. An example of this can be that during development using a waterfall lifecycle any analysis of the system cannot be done until some design of the system has been done, and design of the system cannot begin until some coding for the system has started. This results in the waterfall lifecycle having stages which are dependent on previous stages being complete and accurate, in which this never occurs, rendering this development process inadequate [24]. ROPES gets around the problem of incompleteness by using an iterative development as stated above. The iterative lifecycle deals with incompleteness by using a more representative model with the premise that each lifecycle model is not only planned to execute once but many times, so that each time the model runs it ends in a resulting prototype. Iterative lifecycles are also much more flexible and can easily be manipulated to the size and complexity of the system that is being developed [24].

A disadvantage to using ROPES is that while this methodology is flexible and agile this comes at a cost, as an iterative lifecycle is much more difficult to plan and control and are

most commonly not supported by project management tools. Another disadvantage is that ROPES is usually a development methodology based around teams of developers, this is where the modification comes in as this project is single developer only, so the amount of work per cycle will be scaled down to adjust for only a single developer.

3.4 The Hardware and Software in Use

The only hardware required for this project is access to a computer. As the project is mainly developed on a Linux Operating System the use of a computer running Fedora or any other type of Linux OS would be needed.

The software needed for this project will consist of an Integrated Development Environment (IDE), this project used QT. Along with QT the necessary C++ development libraries will be needed due to language preference and because this project aims to create software which is object orientated, as well as the GNU Scientific Library known as GSL. GSL is used during analysis as it is a C and C++ numerical library with built-in methods and functions capable of performing complex analysis such as measuring the skew, mean, variance and kurtosis of data points allowing for easy output to identify whether the sampled data points are uniformly or normally distributed.

Other software needed for this project is the use of the OpenGL libraries along with glut. These are used in the visualisation program and create a way of interacting with the GPU of a computer to allow for the creation of graphical windows and drawing of data patterns. Another piece of software needed for this project is the use of the boost program options. Boost program options has been used inside of this project to allow for launch configurations to be set on all developed programs. This allows for the quick change in parameters altering which method of distribution and dimensions can be generated, analysed and visualised as rebuilding the program will no longer be necessary.

The final piece of software needed is a form of version control. The version control used for this project was git. Git allows for the creation of a local or online repository, these repositories allows for the saving or uploading of different builds during the development cycle. Having these backups allows for constant integration during the development cycle and creates a safety net as each build can be rolled back if errors or emergencies occur. However due to the project being limited to a single individual git was mainly used for creating backups and storing them in an online repository if a rollback was ever needed.

Chapter 4 – Design and Implementation

Before moving onto the design of the system an analysis is needed to create and define requirements in which the system should undertake and achieve. The first section of analysis is the user requirements which will form basic outlines for the functionality of the developed system, followed by the system requirements which will expand upon the user requirements in more detail. The analysis will then be finished off by looking into scenarios in which the system expects to encounter and the use cases relevant to each scenario.

4.1 User Requirements

The user requirements below give a brief overview on how the system as a whole is intended to work. However due to the automatic nature of the system, there are very few requirements needed.

1. Create distribution patterns based on method, number of pattern points and chosen dimensions.
2. Store generated patterns into a file.
3. Retrieve generated patterns from file.
4. Analyse retrieved patterns to determine characteristics.
5. Visualise two or three dimensional patterns.

4.2 System Requirements

Using the user requirements, system requirements will expand upon them going into further detail breaking each point down into multiple requirements giving a much better understanding into how each requirement can be met by the system.

1. Creating patterns using alternative methods, pattern size and chosen dimension.
 - Patterns can be created by choosing one of three methods. Random, OSPHERE or Uniform Gaussian.
 - Patterns are stored in either vectors, which handle one to three dimensions, while four dimensional patterns are stored in quaternions.
 - Pattern size can be configured before execution of program.
 - Dimensions can be chosen before execution of program. With valid ranges between one and four dimensions.

2. Storing generated patterns to a file.
 - Depending on dimensions used, in the case of one to three dimensions vectors will be written to file in single lines with their X, Y and Z coordinates respectively. If four dimensions are used quaternions will be handled the same way however the S, I, J and K values will be used instead.
 - The file name will be handled automatically and created combining the method of distribution chosen and the number of dimensions.
3. Retrieving patterns from file.
 - The file manager will generate a file name based on method of distribution chosen and the number of dimensions.
 - Data stored in file will be placed into either vector or quaternions lists based on chosen dimensions
4. Analysing characteristics in patterns
 - Based on the number of dimensions the respective list will be analysed by GSL to determine the sampled mean, variance, skew and kurtosis.
5. Visualisation of Patterns
 - Before execution of program the selected method of distribution and dimensions are chosen, with valid ranges of dimensions being limited to two or three dimensions.
 - Chosen file is read and data is stored
 - Using OpenGL and glut the patterns are then displayed to screen.

4.3 Scenarios

This section of the analysis aims to generate scenarios to model real life situations in which the program is expected to encounter.

4.3.1 Scenario 1 - Generating Patterns

During execution of the program three initial parameters are chosen and set. The first parameter has a value range within one and three and is used to set the method of distribution. With one being random distribution, two using the Box-Muller/Uniform Gaussian and finally three being the OSPHERE algorithm. The second parameter is used to set the value of the number of points to be distributed inside of the pattern and the final parameter is used to set the number of dimensions the pattern will take place in.

When all parameters are set patterns will be automatically generated and will populate a list based on the size parameter passed. Upon population of the list a filename will also be automatically generated based on the method and dimensions used and the values stored inside of the generated list will be stored. Lastly the program will display a message on screen informing the completion of pattern generation.

4.3.2 Scenario 2 – Analysis of Patterns

The analysis of patterns begins by taking two parameters during execution. The first parameter choosing which method of distribution is to be analysed, the second parameter detailing the number of dimensions. The process begins by generating a filename based on method and dimensions and then locating that file which was earlier created during generation. Upon location, the file is read, reading in the X, Y and Z values or S, I, J K values for vectors and quaternions respectively. These values are then stored inside of a list which is then passed to GSL in which it can then be analysed. GSL will analyse the data retrieving the mean, variance, skew and kurtosis of the selected data pattern, which will then be displayed on screen.

4.3.3 Scenario 3 – Visualisation of Patterns

To begin the visualisation process two parameters are used during execution. The first parameter is used to select the method of distribution while the second parameter sets the dimensions used. However due to limitations the valid range for dimensions is only between two and three.

Once parameters are set a filename is generated using the method of distribution and dimensions and is located. When located the file is read, reading in the X, Y and Z values and storing them into a list. Upon population of the list an OpenGL object is created and used to display the populated list to screen.

4.4 Use Case Documentation

Table 1: Use case documentation for generating patterns.

Use Case Number: 1	Use Case Name: Generating Patterns
Goal: To generate patterns.	
Description: Generate patterns based on parameters and store them to file.	
Actors: None	
Frequency of Execution: One cycle per execution.	
Scalability: None	
Criticality: Error producing patterns can damage analysis.	
Primary path: The following sequence is carried out whenever generating patterns Parameters passed to set method, pattern size and dimensions. Generation begins and populates list. Filename generated based on method of distribution and dimensions. List values stored to generated file. Display message showing completion of generation.	
Use cases related to primary: None	
Alternatives: None	
Use cases Related to alternatives: None	
Exceptions: If invalid ranges for generation are selected an error message will be displayed.	
Use cases related to exceptions: None	

Table 2: Use case documentation for analysis of patterns.

Use Case Number: 2	Use Case Name: Analyse of Patterns
Goal: To analyse generated patterns to determine characteristics	
Description: Read patterns from file and analyse them to determine mean, variance, skew and kurtosis.	
Actors: None	
Frequency of Execution: One cycle per execution.	
Scalability: None	
Criticality: Error producing patterns can damage analysis.	
Primary path: The following sequence is carried out whenever analysing patterns Parameters passed to set method and dimensions. Filename generated based on parameters. File is read storing values into a list. Values inside of the populated list are analysed. Display characteristics of data to screen.	
Use cases related to primary: None	
Alternatives: None	
Use cases Related to alternatives: None	
Exceptions: If invalid ranges for analysis are selected an error message will be displayed.	
Use cases related to exceptions: None	

Table 3: Use case documentation for visualisation of patterns.

Use Case Number: 3	Use Case Name: Visualisation of Patterns
Goal: To visualise generated patterns.	
Description: Read patterns from file and display them on screen.	
Actors: None	
Frequency of Execution: One cycle per execution.	
Scalability: None	
Criticality: Limitations only allow for visualisation in two or three dimensions.	
Primary path: The following sequence is carried out whenever visualising patterns Parameters passed to set method and dimensions. Filename generated based on passed parameters. File is read and populates a list with values read in. List values are displayed on screen.	
Use cases related to primary: None	
Alternatives: None	
Use cases Related to alternatives: None	
Exceptions: If invalid ranges for method and dimensions are selected an error will be thrown.	
Use cases related to exceptions: None	

4.4.1 Use case Diagrams

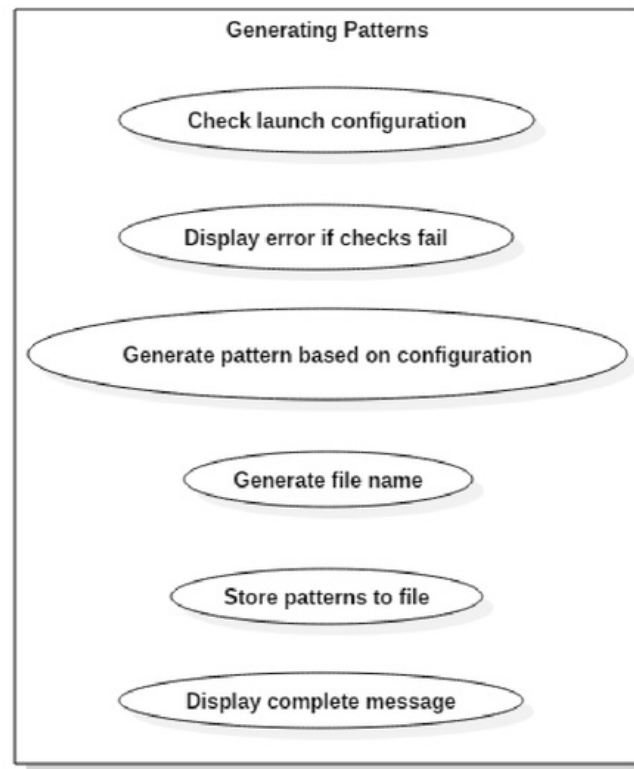


Figure 6: Use case diagram for generating patterns.

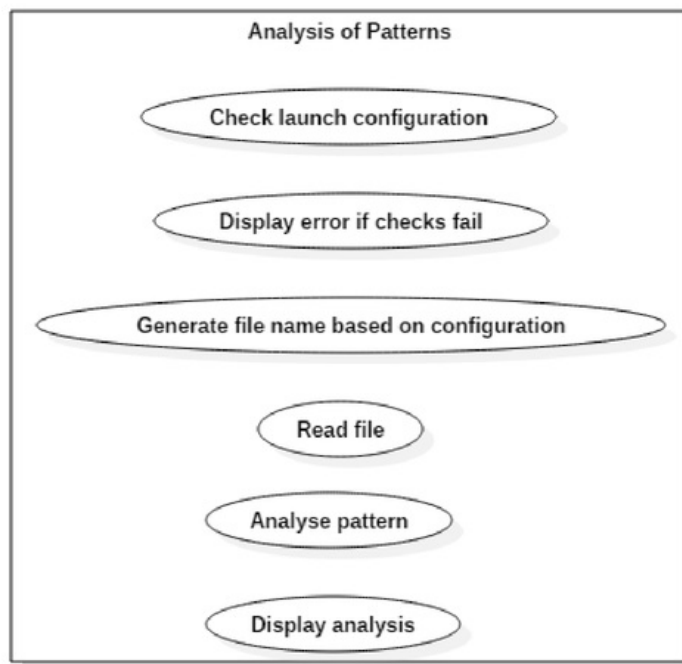


Figure 7: Use case diagram for analysing patterns.

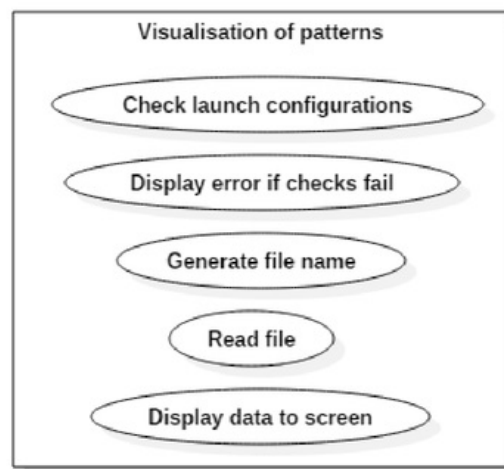


Figure 8: Use case diagram for visualisation of patterns.

4.5 Design

Once analysis of the system had been outlined and requirements had been refined the next step was the design of the system. Following the earlier mentioned development methodology of ROPES there are multiple stages of design. The first layer of design is architectural which looks at the logical and physical details of the developed system and how they work together. The second stage of design is mechanistic which will contain sequence diagrams detailing how the system runs for generating patterns, analysing generated patterns and finally showing two or three dimensional visualisations of patterns. The last section of design focuses on detail which will provide activity and state diagrams of the system.

Following the ROPES methodology which relies on a spiral approach to development the designs of the system had multiple iterations during development. The diagrams below represent the finalised versions of each section of design.

4.5.1 Architectural

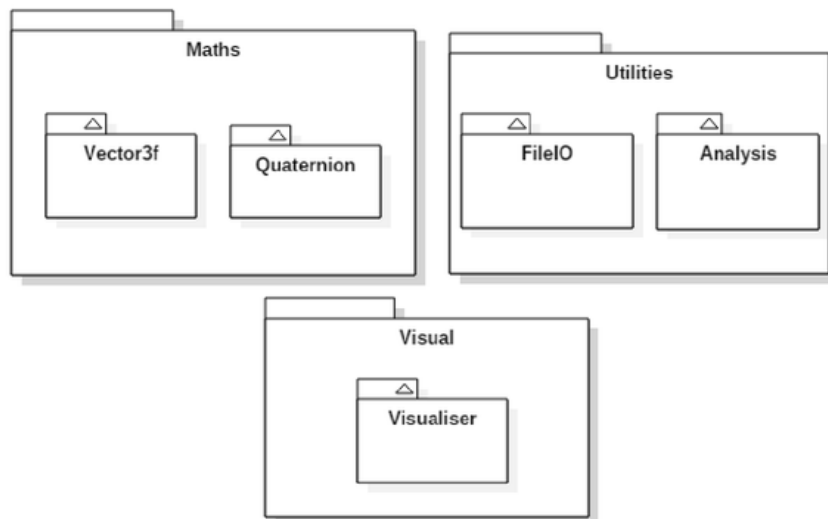


Figure 9: Package diagram for the completed system.

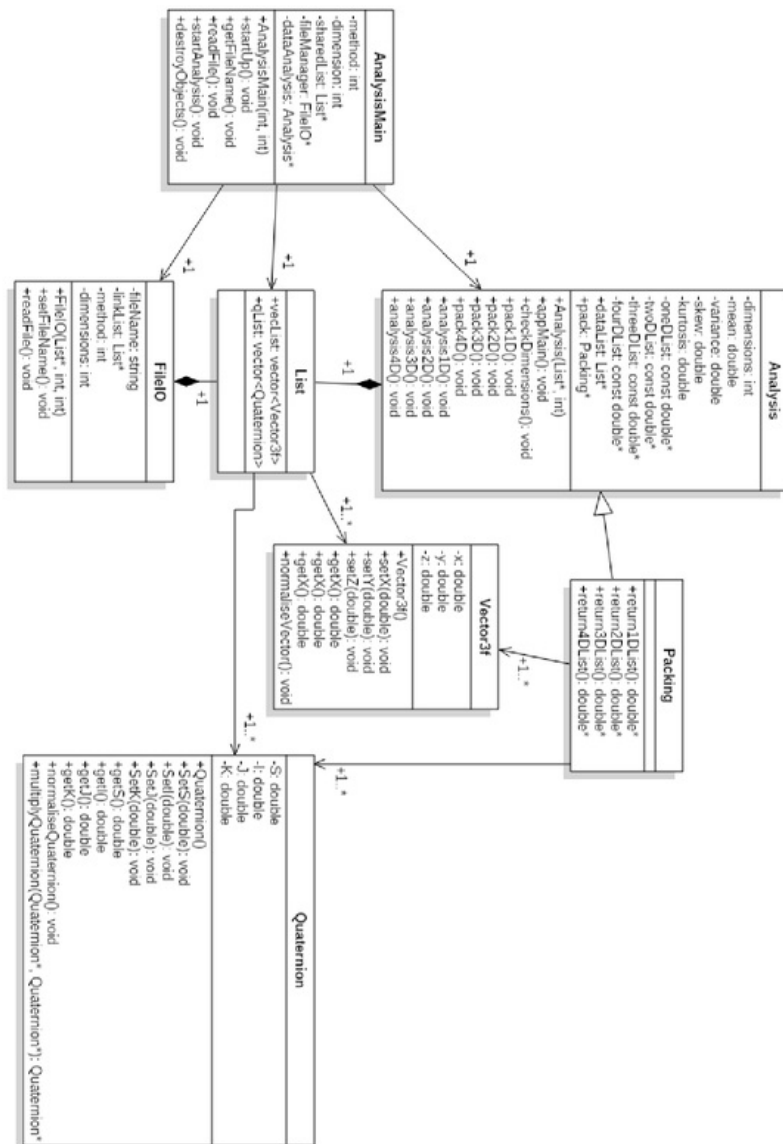


Figure 11: Class diagram for analysis of points.

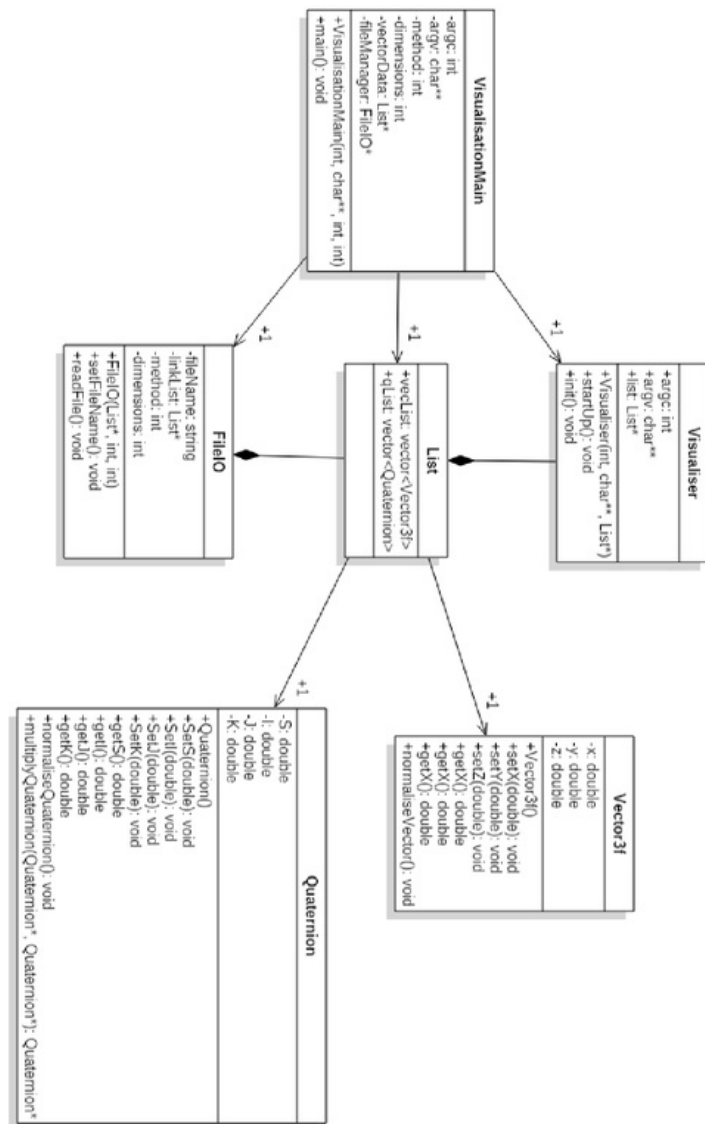


Figure 12: Class diagram for visualisation of points.

4.5.2 Mechanistic

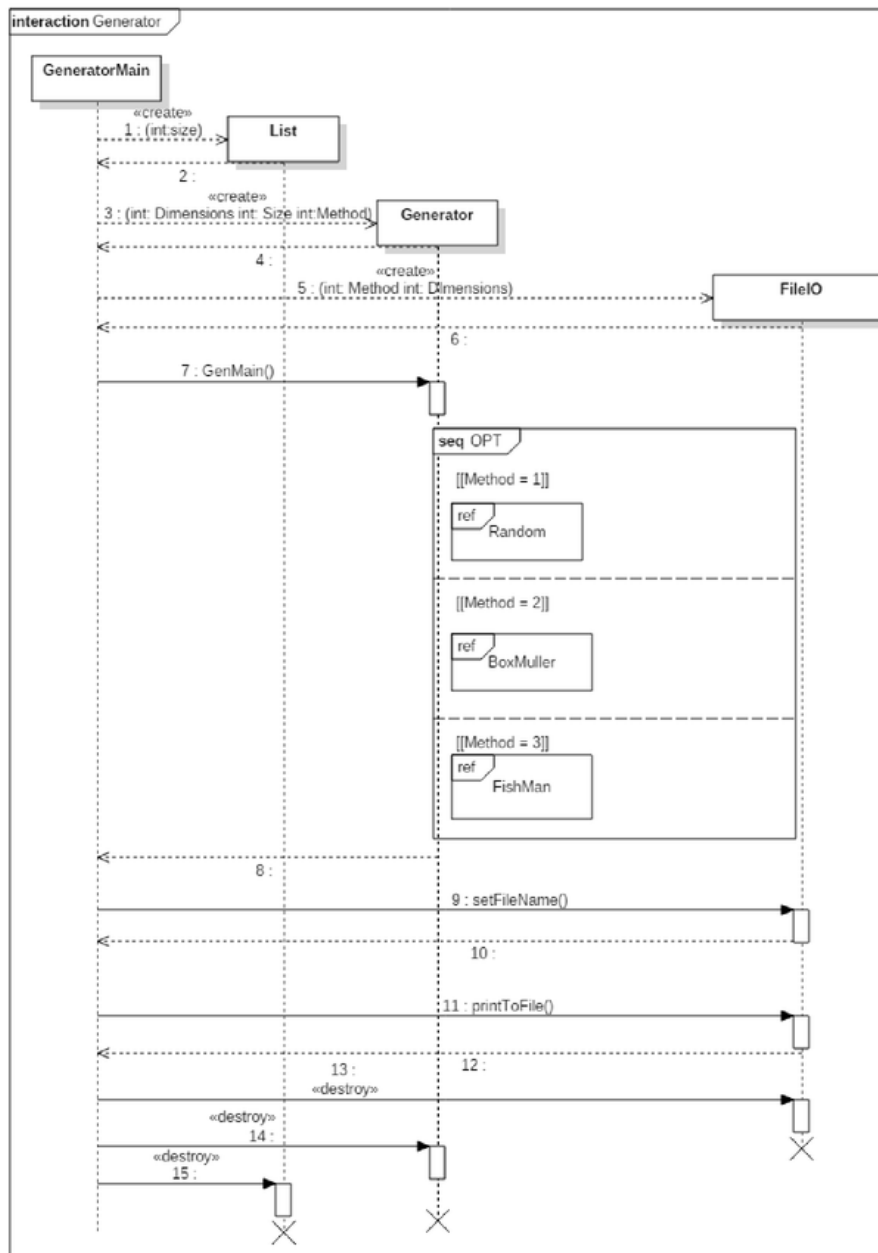


Figure 13: Sequence diagram for generation.

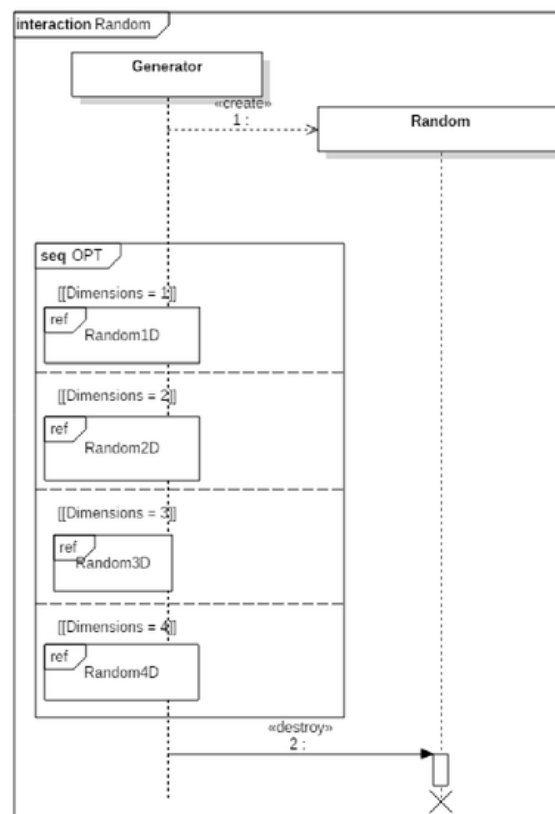


Figure 14: Ref for random method in generation sequence.

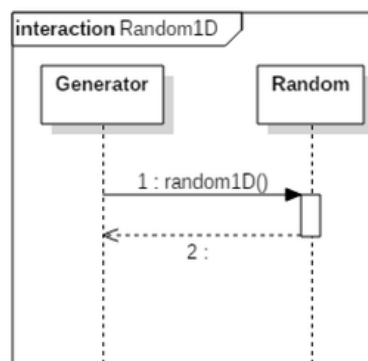


Figure 15: Random1D Ref.

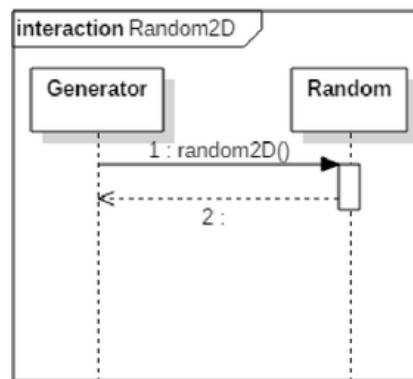


Figure 16: Random2D ref.

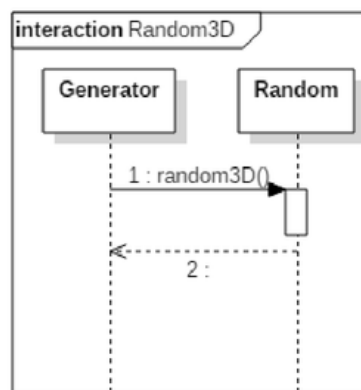


Figure 17:Random3D ref.

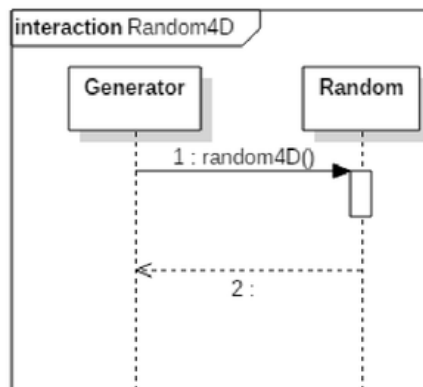


Figure 18: Random4D ref.

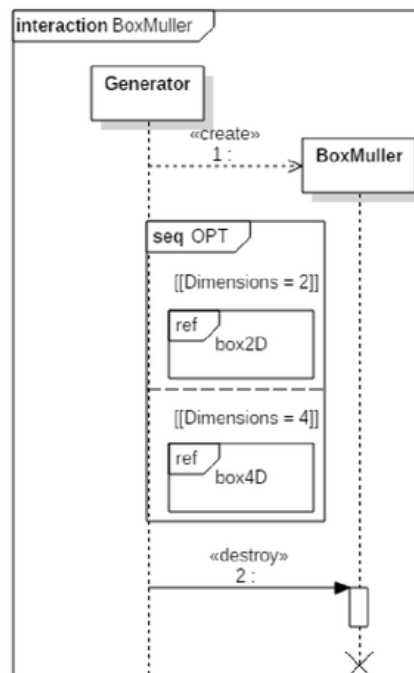


Figure 19: Box-Muller ref for generation sequence.

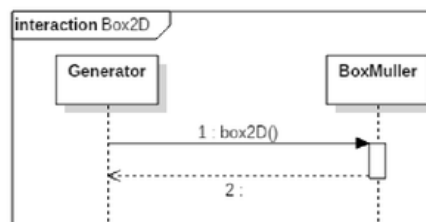


Figure 20: Box2D ref.

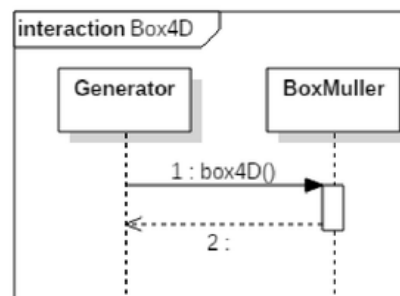


Figure 21: Box4D ref.

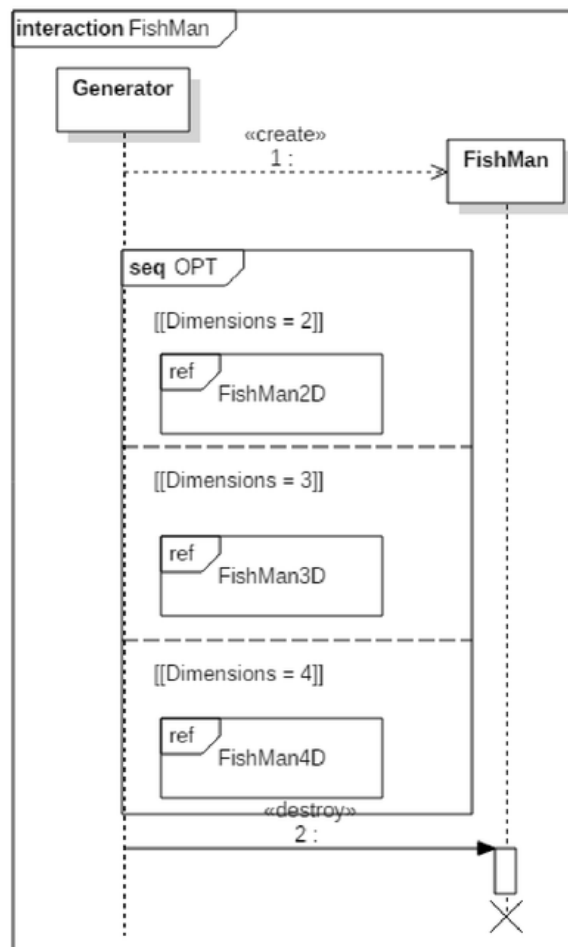


Figure 22: Fishman red for generation sequence.

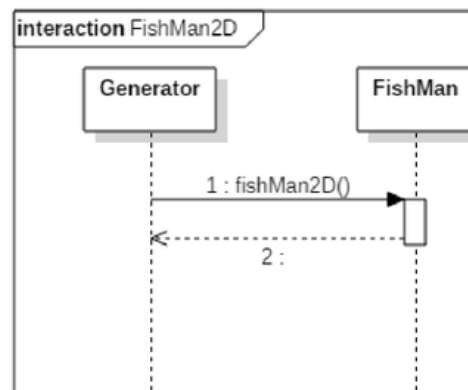


Figure 23: Fishman2D ref.

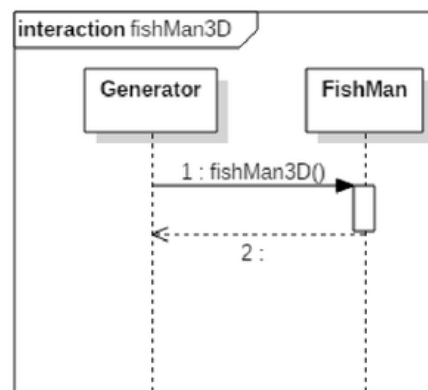


Figure 24:Fishman3D ref.

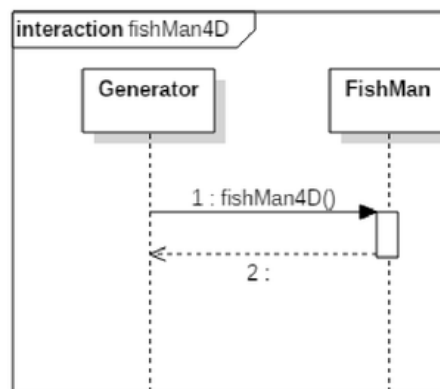


Figure 25: Fishman4D ref.

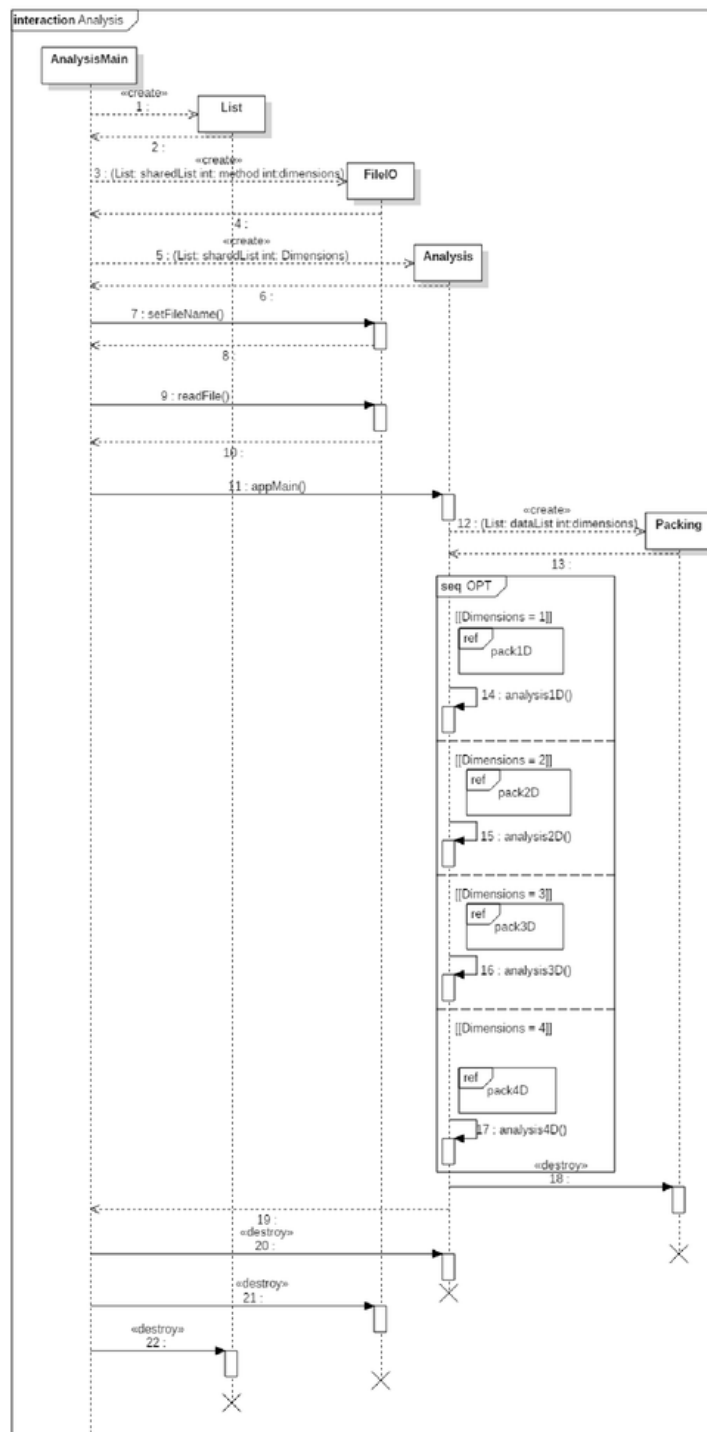


Figure 26: Analysis sequence diagram.

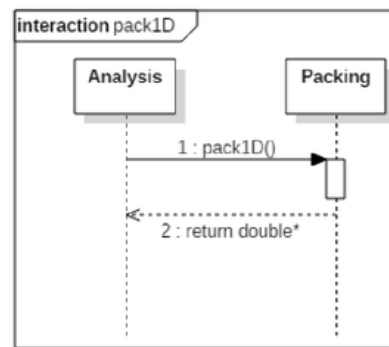


Figure 27: pack1D ref.

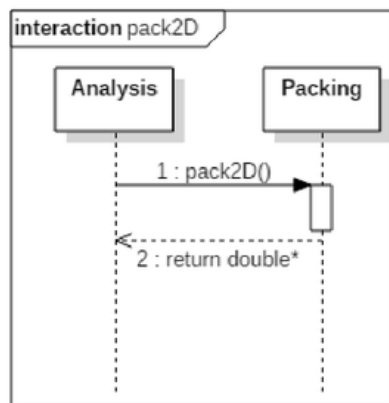


Figure 28: pack2D ref.

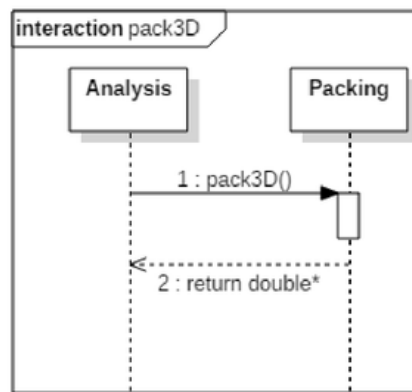


Figure 29: pack3D ref.

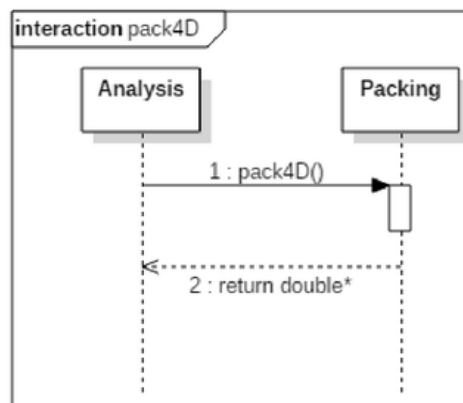


Figure 30: pack4D ref.

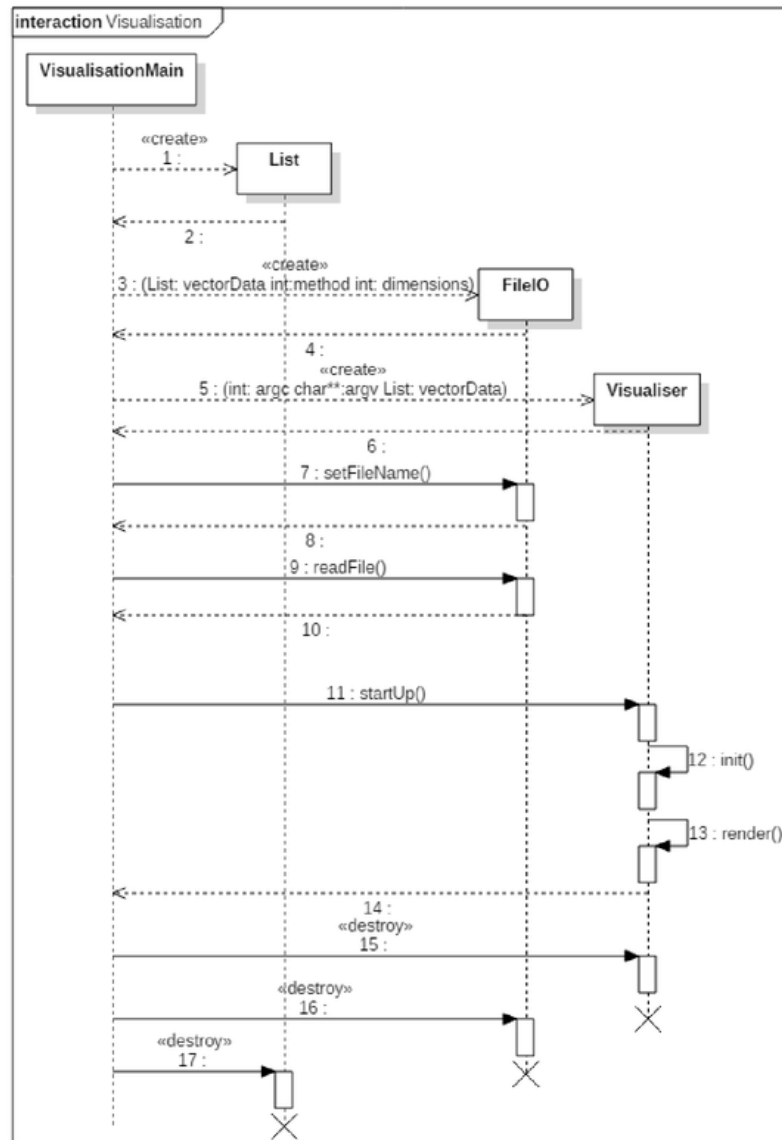


Figure 31: Visualisation sequence diagram.

4.5.3 Detail

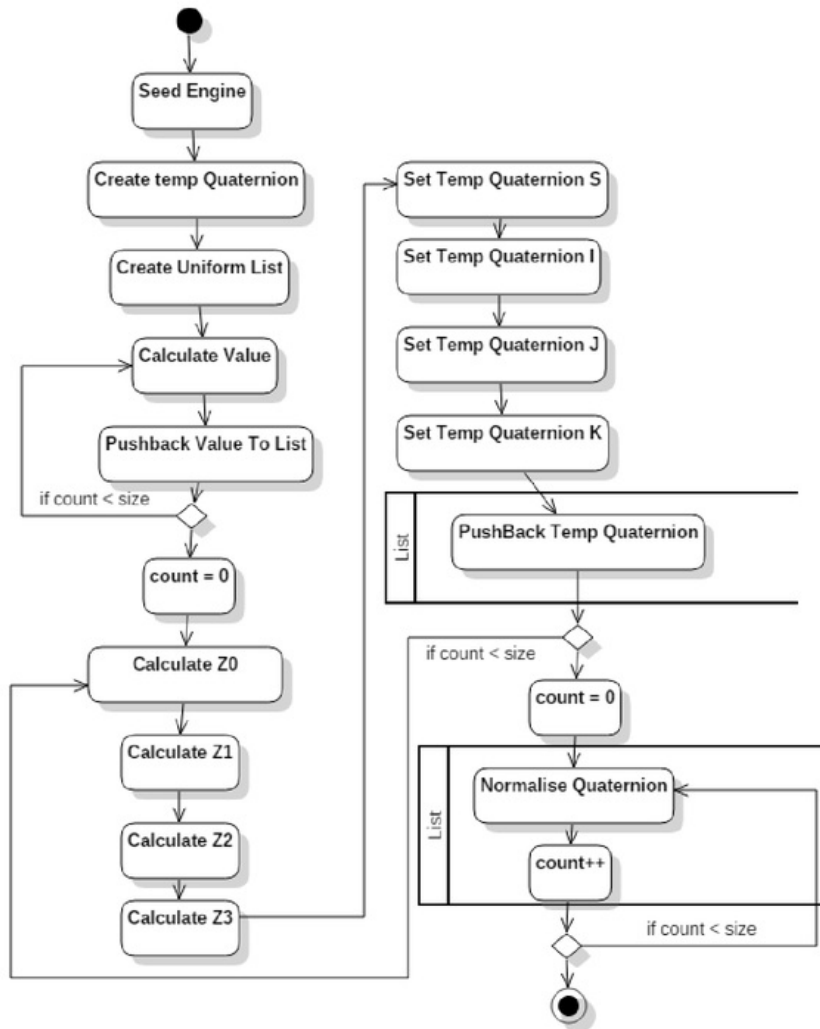


Figure 32: Activity diagram for Box-Muller 4D.

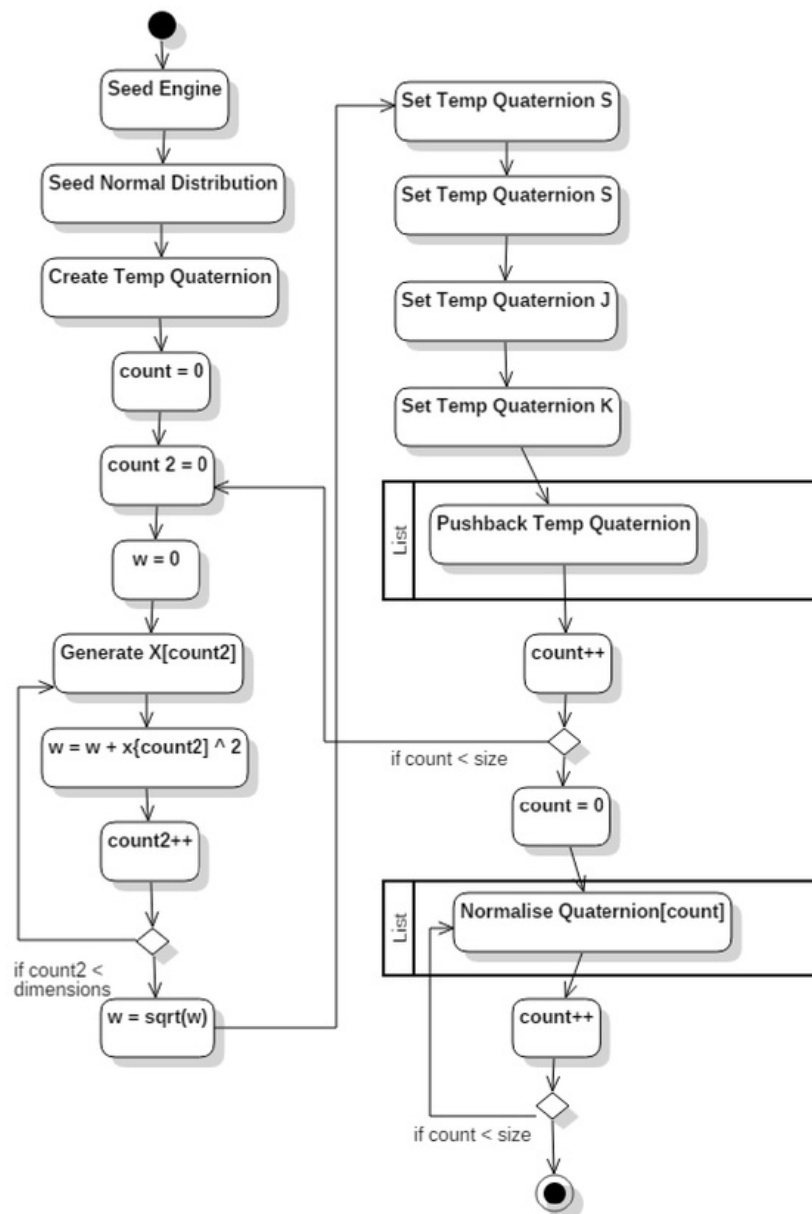


Figure 33: Activity diagram for Fishmans OSPHERE 4D.

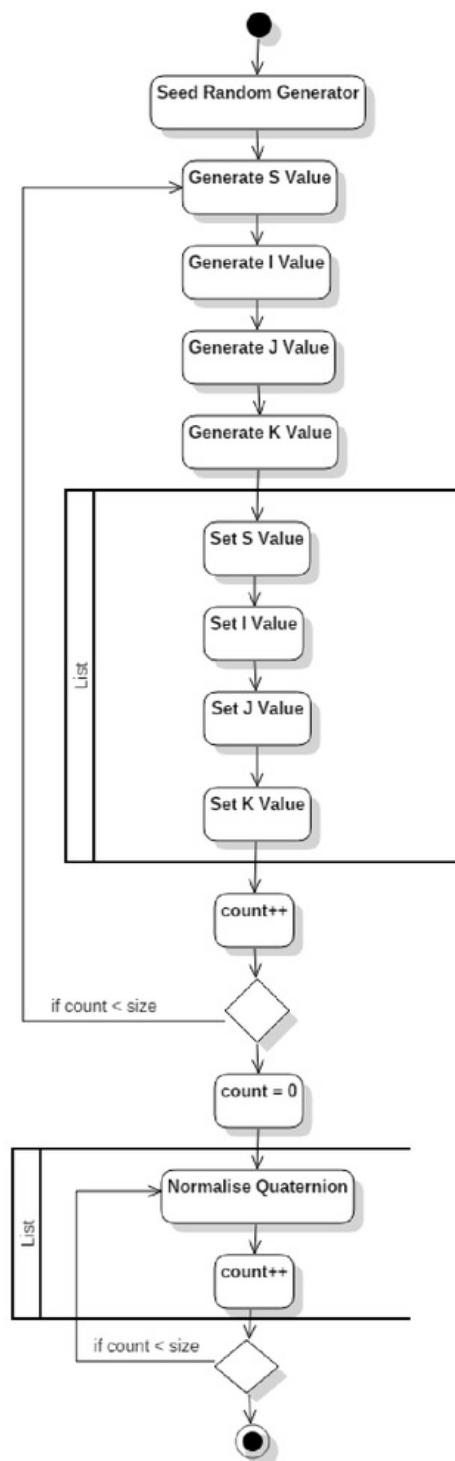


Figure 34: Activity diagram for random 4D.

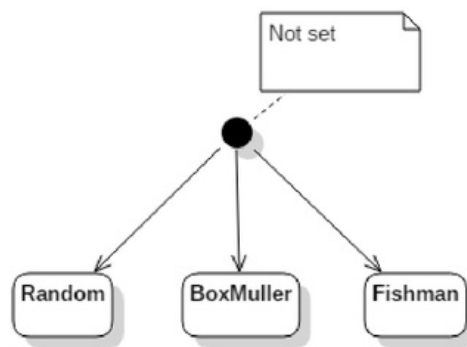


Figure 35: State diagram showing states all the programs can enter.

4.6 Implementation

Following the ROPES development methodology each system developed was built around the spiral approach which uses an iterative development method. Using the spiral approach allowed for the creation of prototypes, prototyping was useful in the implementation as it allowed for additions on previous prototypes which in the end lowered the amount of time and dedication needed to finish the developed systems.

The implementation stage for this section has three spirals of development. The first spiral relates to the development of the generator program in which the main objective is to distribute points in multidimensional space using alternative methods. The second spiral is the development of the analysis program. This programs main objective is to read in values from a file related to a distribution pattern and dimension and analyse the patterns and display the characteristics such as the skew and kurtosis. The final spiral is for the visualisation program, used to visualise the patterns that have been distributed in two or three dimensions.

Beginning with the generation program, the initial iteration consisted of nine separate programs. These programs were the three distribution methods chosen, these being random, Fishmans OSPHERE and uniform Gaussian. Each method was separated by dimensions from one to three, leading to the development of the nine initial programs. During this iteration only one class per program existed, that class being a vector, due to distribution methods currently only working in one to three dimensions. Below are figures showing code examples of each method of distribution.

```

Vector3f tempVec;
double randomValue;
for(int i = 0; i < randomList.size(); i++)
{
    randomValue = (randMax - randMin) * (((double)rand()) / (double) RAND_MAX)) + randMin;
    tempVec.setX(randomValue);
    randomList[i] = tempVec;
}

```

Figure 36: Random distribution in one dimension code.

```

Vector3f temp;
double val = 0;
double z0, z1;

unsigned seed = std::chrono::system_clock::now().time_since_epoch().count();
std::vector<double> uniformList;
std::default_random_engine generator(seed);
std::uniform_real_distribution<double> distribution(0.0, 1.0);

for(int i = 0; i < boxList.size() * 2; i++)
{
    val = distribution(generator);
    uniformList.push_back(val);
}

for(int i = 0; i < boxList.size(); i++)
{
    z0 = sqrt(-2.0*log(uniformList[i*2])) * cos((2*M_PI) * uniformList[(i*2)+1]);
    z1 = sqrt(-2.0 * log(uniformList[i*2])) * sin((2 * M_PI) * uniformList[(i*2)+1]);

    temp.setX(z0);
    temp.setY(z1);

    boxList[i] = temp;
}

for(int i = 0; i < boxList.size(); i++)
{
    boxList[i].normaliseVector();
}

```

Figure 37: Box-Muller code for one dimension.

```

unsigned seed = std::chrono::system_clock::now().time_since_epoch().count();
std::default_random_engine generator(seed);
std::uniform_real_distribution<double> d(0.0, 1.0);

double X[2];
Vector3f tempVec;

int j;
for(j = 0; j < fishManList.size(); j++)
{
    int i = 0;
    float w = 0;

    while(i <= 1)
    {
        X[i] = d(generator);

        w = w + pow(X[i], 2);

        i++;
    }

    w = sqrt(w);
    i = 1;

    tempVec.setX((1*X[0])/w);
    tempVec.setY((1*X[1])/w);

    fishManList[j] = tempVec;
}

for(int i = 0; i < fishManList.size(); i++)
{
    fishManList[i].normaliseVector();
}

```

Figure 38: Fishmans OSPHERE algorithm in two dimensions.

The purpose to this iteration was to develop an understanding as to how these methods worked in their lower dimensions before moving them up to four dimensions. A test harness was created for this iteration and can be seen in table 4, running unit tests on the vector class.

Once an understanding of the distribution methods was founded the second iteration of development began. This iteration focused on developing a more compact and well-designed program. During this development phase the nine initial programs had now been condensed to three. These three programs consisted of each distribution method, however now dimensional selection was built into each program separately. This development phase also added the classes which were List and FileIO. List is designed to create an array based on a passed size, in which this array will hold a list of vectors, while FileIO is created to allow manipulation of data to be stored and is created based on method of distribution and dimension. FileIO has a special relationship with List as it stores an internal reference to List, allowing for quick manipulation of Lists data through FileIO while reading. Integration tests can be seen for this iteration in table 5.

The final iteration for the generation program consisted of placing all three distribution methods into a single program allowing for states to be set to choose the desired method

of distribution and dimensions. During this iterations a quaternion class was also added allowing for the distribution to extend into four dimensions, unit tests for the quaternion class can be seen in table 6. The final development phase of the generation program added the boost command line parser. Using this, launch configurations could be set before execution of the program. Each parsed parameter is checked before running and will display an error if checks fail. The final design of this program after multiple iterations and changes can be seen in figure 10 and 13.

The second program to be implemented was the analysis. This program consisted of multiple classes used previously in the generator program, due to this the development cycle was much shorter and precise as testing had already been completed and functionality of the classes was already known to work. The first iteration began by taking the FileIO, List, Vector and Quaternion classes from the generator method. During this phase of development the focus was on creating a List object whose arrays could be populated by FileIO by reading values from a .txt file containing data patterns.

Once satisfied with the data of files being read correctly, the second iteration began. This iteration focused on the addition of two new classes, the analysis class and packing class. The analysis class handles the GSL object whose purpose is to analyse sampled data and return characteristics, in this case looking for the mean, variance, skew and kurtosis. However GSL requires patterns to be placed in const double arrays before processing, this is where the packing class is used as it will return an array based on dimensions back to analysis so that it can be processed by GSL. The final implementation of the analysis was the introduction of boost command line parser allowing for the use of launch configurations. Final designs for this program can be seen in figure 11 and 26.

The last spiral involved the implementation of the visualisation component. During the first iteration, the focus was on being able to display data to screen so that points could be visualised. This was achieved by using the earlier implemented Vector and List classes. Using these classes a list of vectors could be generated manually. Once a list was populated a visualisation class was added which holds the OpenGL and glut libraries used for creating windows and graphics. The vector list currently held by List was passed by reference to visualisation so that patterns could easily be extracted and displayed to screen.

When patterns were capable of being displayed from manual creation the final iteration began. This iteration replaced manual pattern distribution with automatic by implementing the FileIO class, capable of reading patterns from file. The FileIO class would

populate the array held in List, which the visualisation class could access and display those patterns to screen. The last part of implementation to this section was the introduction of the boost command line parser allowing for launch configurations to be set. Below is an example of the visualisation component in use. The related final class and sequence diagrams can be found in figure 12 and 31.

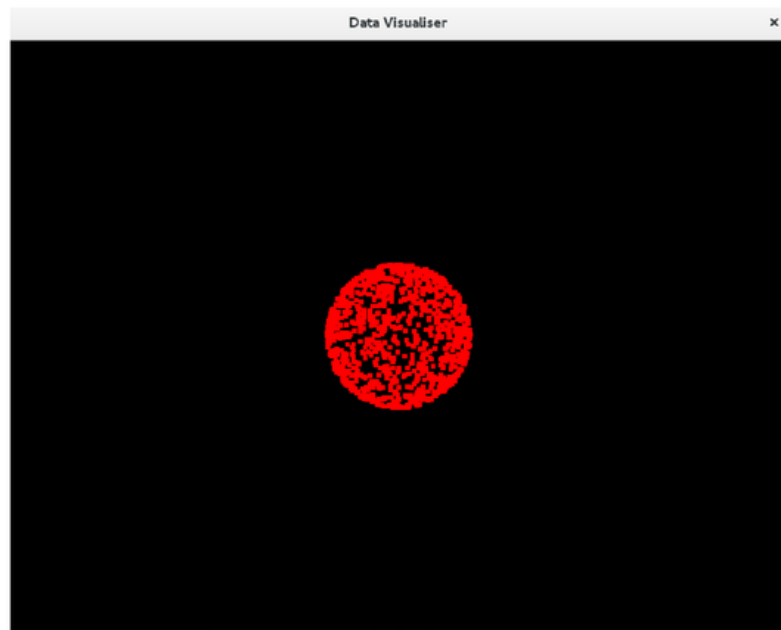


Figure 39: 3D visualisation of Fishmans OSPHERE

4.6.1 Testing

4.6.1.1 Unit Testing – Vector Class

Table 4: Unit tests of vector class.

Test Number	Description	Input	Expected outcome	Actual outcome	Success/Fail
1	Testing the creation of a new vector object	N/A	A new vector is created with the values of 0,0,0	A vector was created storing the values 0,0,0	Success
2	Testing the setX method	1	The newly created vectors X value becomes 1.	Vectors X value changed to 1.	Success
3	Testing the setY method	2	The newly created vectors Y value becomes 2.	Vectors Y value changed to 2.	Success
4	Testing the setZ method	3	The newly created vectors Z value becomes 3.	Vectors Z value changed to 3.	Success
5	Testing the getX method	N/A	The vectors X value is returned.	Vector returned 1.	Success
6	Testing the getY method	N/A	The vectors Y value is returned.	Vector returned 2.	Success
7	Testing the getZ method	N/A	The vectors Z value is returned.	Vector returned 3.	Success

4.6.1.2 – Generator Integration Tests

Table 5: Integration tests for list and vector.

Test Number	Description	Input	Expected outcome	Actual outcome	Success/Fail
1	Creation of a list object.	Size = 10	A new list object will be created based on passed size.	A new list object was created with 10 vectors	Success
2	Creation of a list object.	Size = -5	Error message will be displayed	Error message was displayed.	Success
3	Creation of a FileIO object.	List = testList Method = 1 Dimensions = 1	A file will be generated with a name random2D.txt, and a reference to list	A file was created with a reference to list and a name of random2D.txt	Success
4	Creation of a FileIO object without a matching method.	List = testList Method = 3 Dimensions = 1	An error message should display as OSPHERE does not work in one dimension.	An error was shown showing invalid selection	Success

4.6.1.3 Quaternion Unit Tests

Table 6: Unit tests for quaternion class.

Test Number	Description	Input	Expected outcome	Actual outcome	Success/Fail
1	Testing creation of a quaternion object.	N/A	A new quaternion is created with values 0,0,0,0	A quaternion was created with values 0,0,0,0	Success
2	Testing the setS method.	1	Quaternion S value set to 1.	S value changed to 1.	Success
3	Testing the setS method.	2	Quaternion S value set to 2.	I value changed to 2.	Success
4	Testing the setS method.	3	Quaternion S value set to 3.	J value changed to 3.	Success
5	Testing the setS method.	4	Quaternion S value set to 4.	K value changed to 4.	Success
6	Testing the getS method.	N/A	Return of 1.	Returned value of 1.	Success
7	Testing the getI method.	N/A	Return of 2.	Returned value of 2.	Success
8	Testing the getJ method.	N/A	Return of 3.	Returned value of 3.	Success
9	Testing the getK method.	N/a	Return of 4.	Returned value of 4.	Success

4.6.3 System Tests

System tests were conducted by testing the interaction between the generation program and the analysis program. Multiple tests per dimension and method of distribution were run and can be viewed in the dimensional tests document in the attached CD. The next section covers the results gathered when running those tests.

4.6.4 Results

This section covers the results gathered. The figures below detail the tests undertaken in one, two, three and four dimensions for each distribution method available. Each test was conducted ten times in each dimension, with one to three dimensions having 1000 test patterns while four dimensions had 625 test patterns due to limitations in the GSL software.

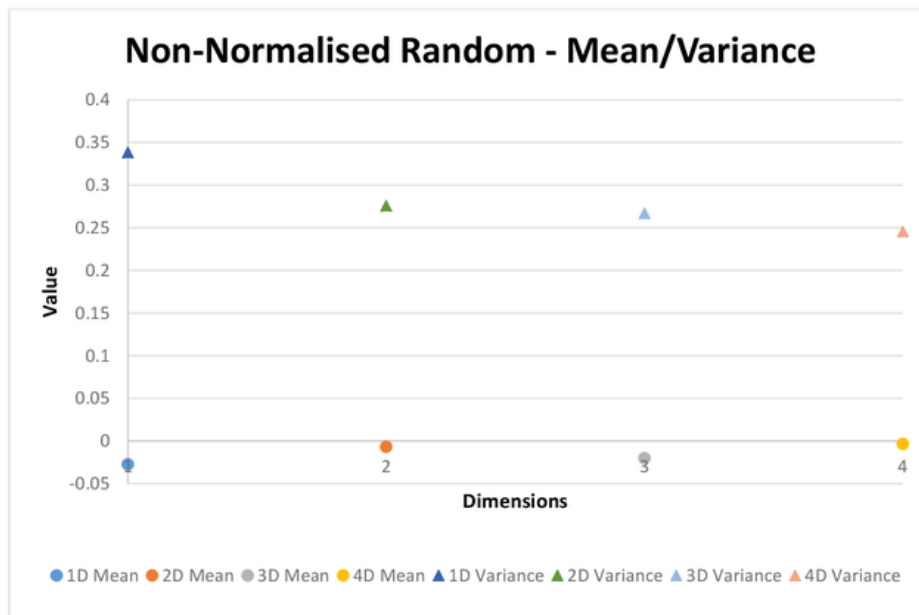


Figure 40: Mean and variance of non-normalised random distribution.

Figure 40 shows the results gathered when conducting tests on non-normalised random distribution in one to four dimensions. From the graph it can be seen that while the number of dimensions increases the variance in the pattern decreases. In one dimensional space the average of variance was 0.3312277 whereas in four dimensions the average had decreased down to 0.255133. It can also be seen that when increasing in dimensions the mean fluctuates between almost zero and below zero, depending on whether or not the dimensions are even or odd.

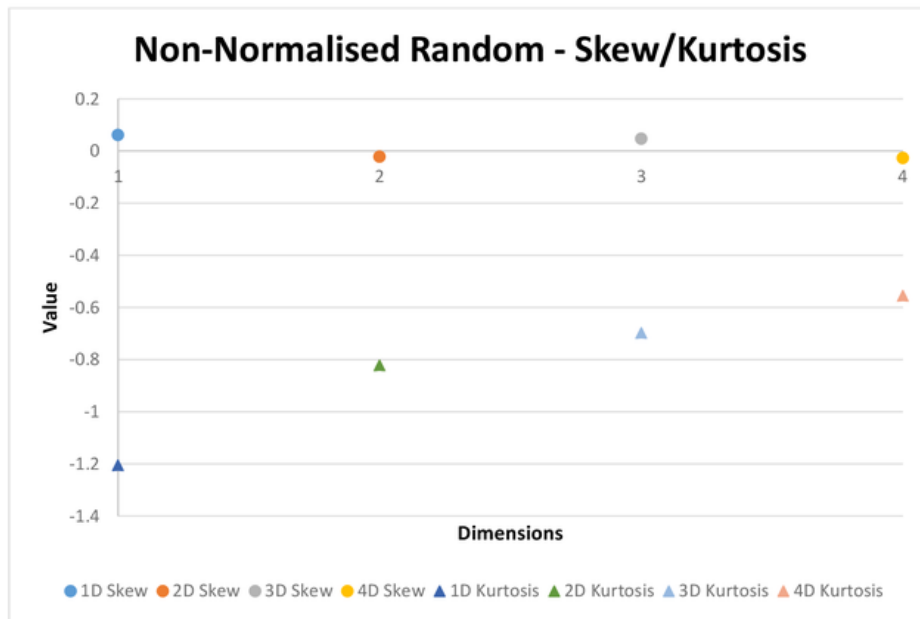


Figure 41: Skew and kurtosis of non-normalised random distribution.

Figure 41 shows the results gathered for the skew and kurtosis of patterns in non-normalised random distribution. Shown in the graph it is clear that the number of dimensions has a linear effect on the kurtosis of patterns as in one dimensions the kurtosis is an average of -1.200702 where in four dimensions that average is now -0.652495. However the mean of the patterns no matter the dimensions will always tend towards zero.

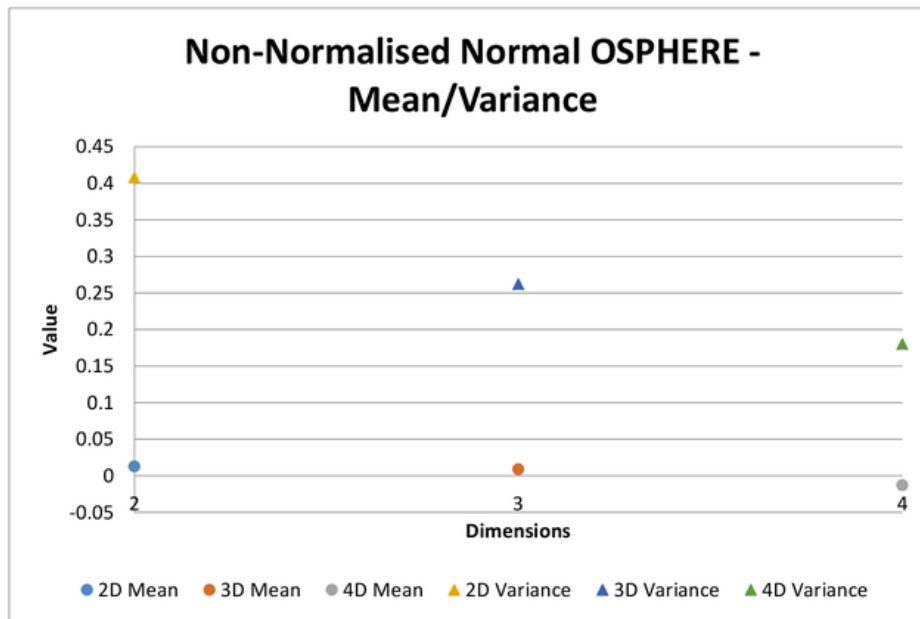


Figure 42: Mean and variance from non-normalised OSPHERE.

Figure 42 shows the results that were gathered when testing non-normalised Fishmans OSPHERE algorithm. Due to limitations this algorithm can only be run in dimensions higher than one, as it is designed to distributed points on a circle or sphere, which is not possible in one dimension. The results show that as the number of dimensions increased the variance inside of patterns decreased with an average of 0.4100045 in two dimensions and an average of 0.190898 in four dimensions, showing that the variance had decreased by half in just two dimensions.

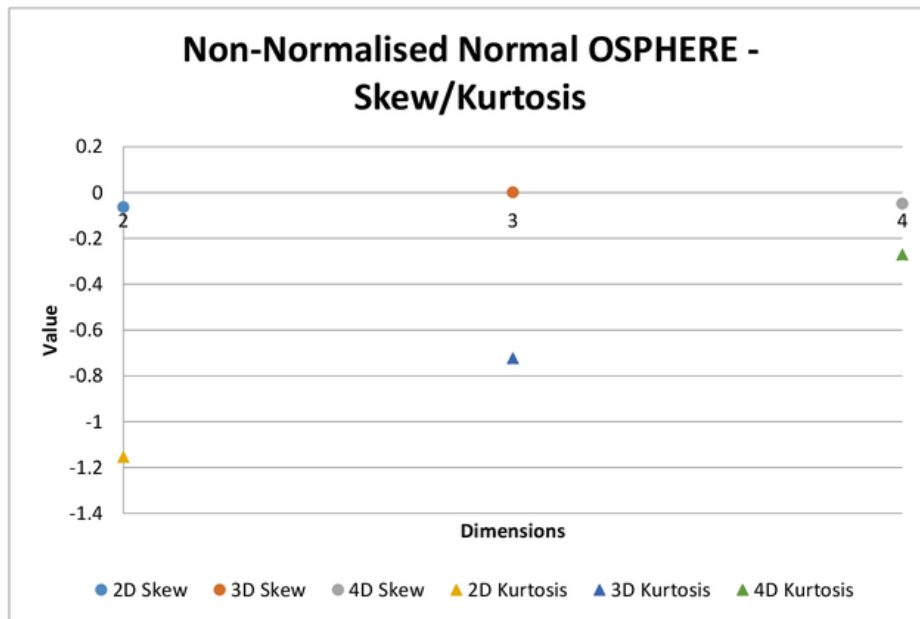


Figure 43: Skew and kurtosis of non-normalised OSPHERE.

Figure 43 shows the skew and kurtosis from tests conducted on the OSPHERE algorithm. The graph indicates that as dimensions increase the skew of the sampled data stays around zero with an average of 0.008457 in one dimension and an average of -0.0118133 in four dimensions, however as the dimensions of the data increases the kurtosis of the sampled points increases linearly with an average of -1.180245 in two dimension, increasing to an average of -0.386495 in four dimensions.

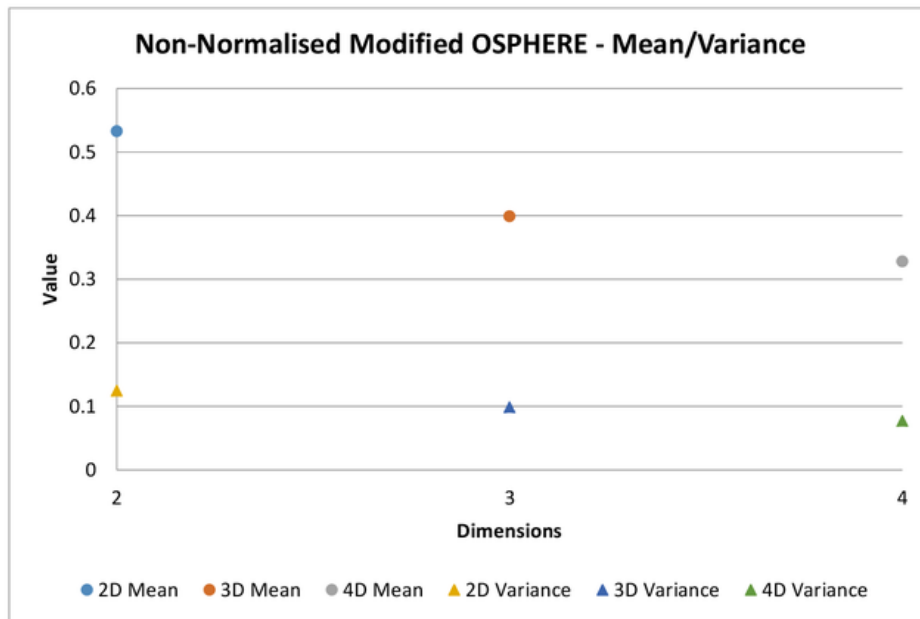


Figure 44: Mean and variance of a modified non-normalised OSPHERE.

Figure 44 shows the results gathered for the mean and variance of a modified OPSHERE algorithm. The algorithm was modified by changing the normal distribution inside of the function to that of a uniform real distribution. The graph shows a high mean and a low variance in patterns, as the number of dimensions increases the mean inside patterns decreases starting from an average of 0.5385554 in two dimensions down to 0.325135 in four dimensions.

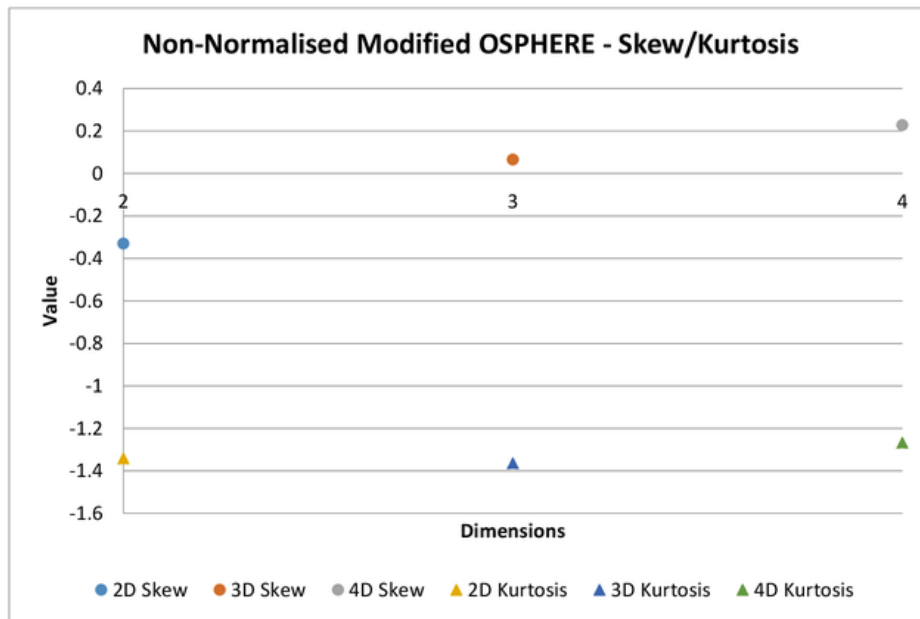


Figure 45: Skew and kurtosis of a non-normalised modified OSPHERE.

Figure 45 shows the skew and kurtosis of patterns gathered when using the same modified OSPHERE algorithm. The graph shows that while dimensions increase from two to four the kurtosis of patterns stays around the same level, however the skew of the patterns increases linearly as the dimensions increase starting from an average of -0.3447482 in two dimensions, finishing off at an average of 0.2428984 in four dimensions.

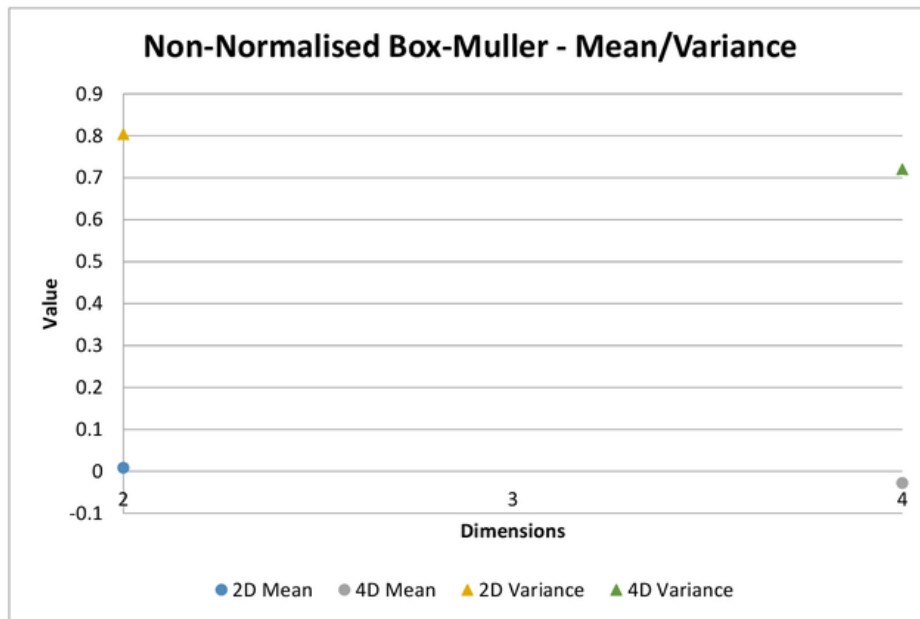


Figure 46: Mean and variance of non-normalised Box-Muller.

Figure 46 shows the mean and variance of patterns in two and four dimensions when tested with a non-normalised Box-Muller approach, as Box-Muller creates pairs of distributed points this method will only work in dimensions of an even number. For these tests that was two and four dimensions. The results show that as the dimensions increase there is a small decrease in the amount of variability in the patterns. This is linked with a small decrease in the mean.

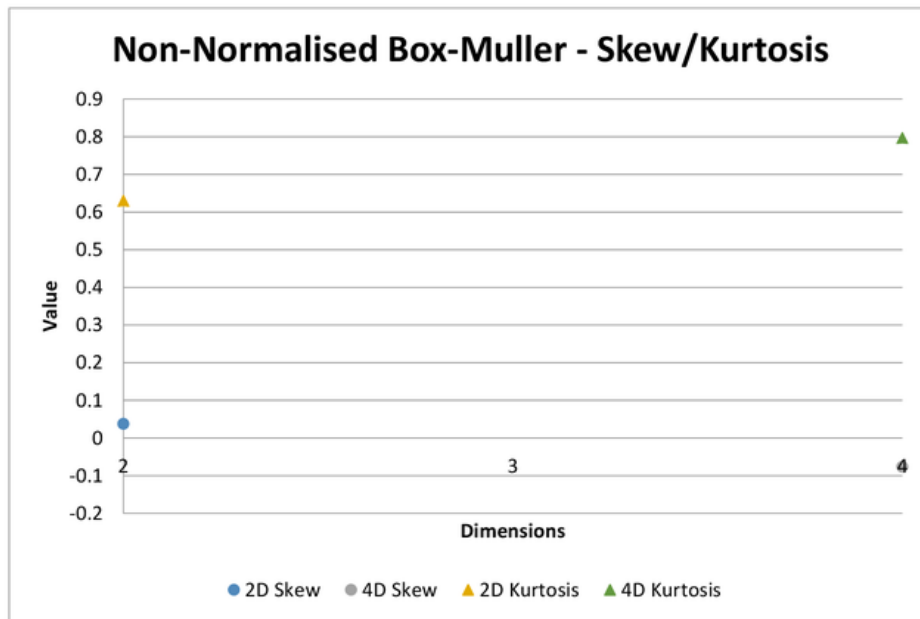


Figure 47: Skew and kurtosis of non-normalised Box-Muller.

Figure 47 shows the skew and kurtosis gathered when testing non-normalised Box-Muller. The graph shows that going from two to four dimensions will increase the kurtosis of the pattern, however the increase in dimensions lowers the skew going from an average skew of 0.024214429 in two dimensions to -0.00885718 in four dimensions.

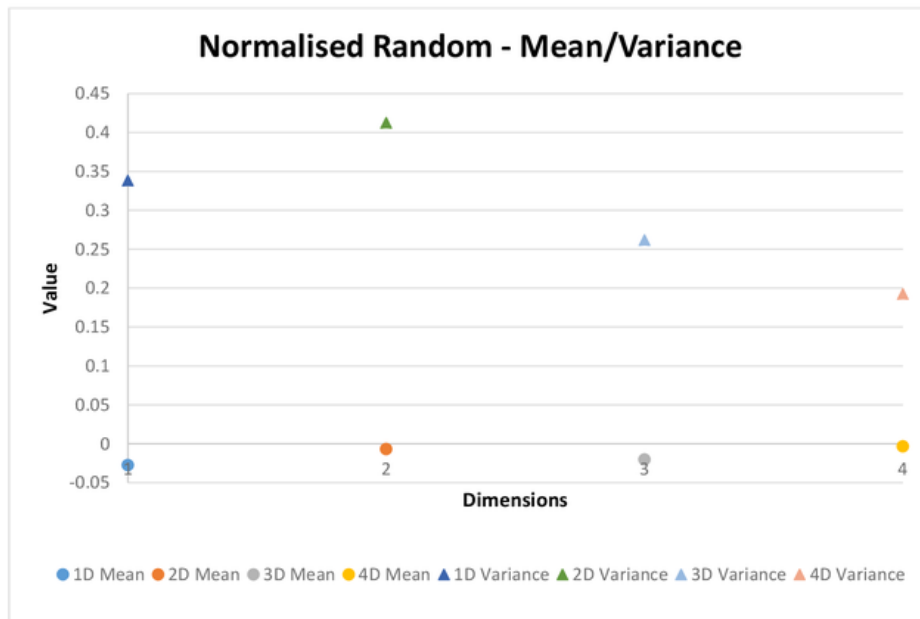


Figure 48: Mean and variance of normalised random distribution.

Figure 48 shows the mean and variance of normalised random distribution through one to four dimensional space. From the graph it is indicated that as dimensions increase the mean of the patterns stays around zero. However as the dimensions increase the variance of the patterns peaks at two dimensions then begins to decrease as the dimensions continue.

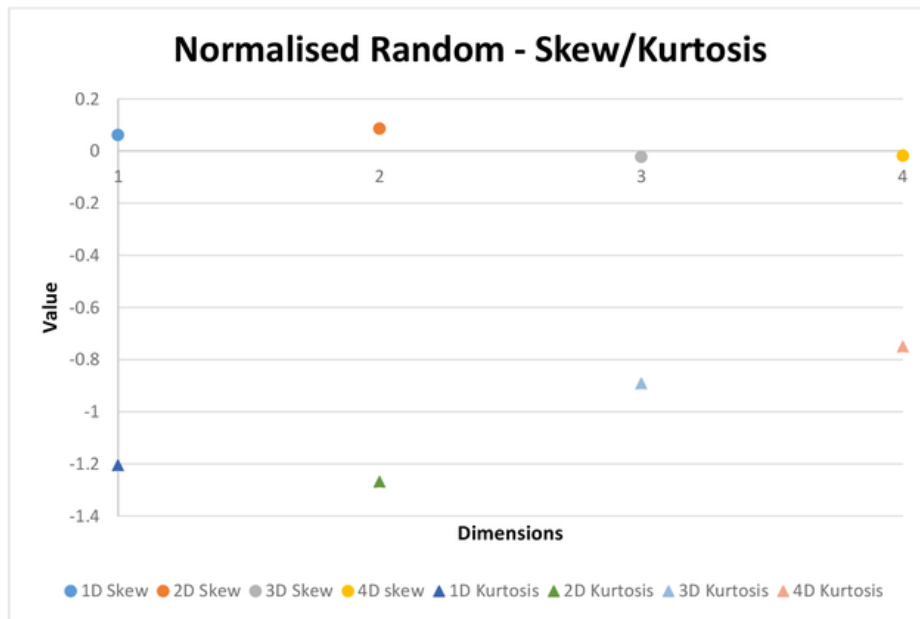


Figure 49: Skew and kurtosis of normalised random distribution.

Figure 49 shows the skew and kurtosis of a normalised random distribution throughout one to four dimensional space. The graph indicates that as dimensions increase the skew of the patterns decreases and the kurtosis increases from an average of -1.200702 to -0.755344.

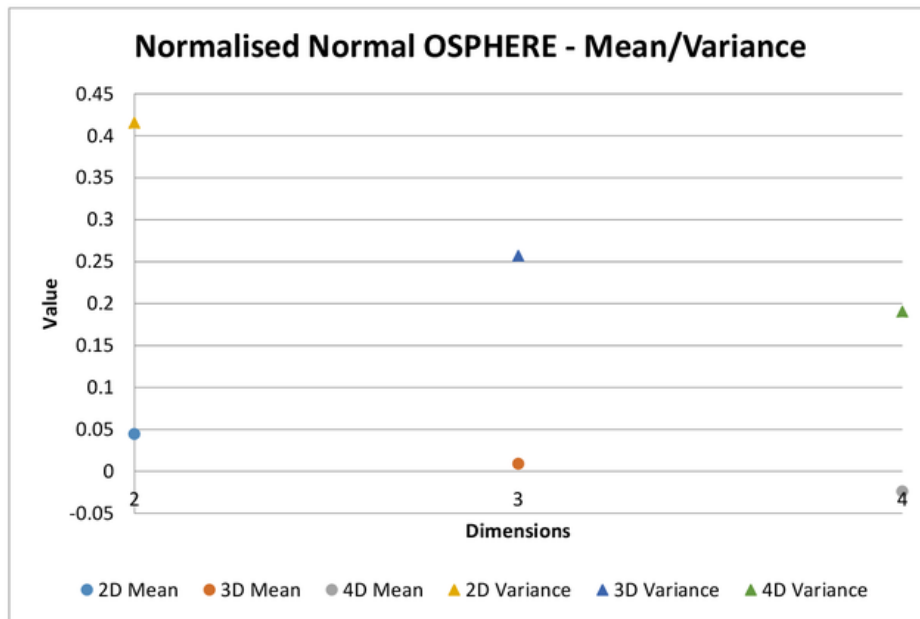


Figure 50: Mean and variance of normalised OSPHERE.

Figure 50 shows the mean and variance of the OSPHERE algorithm using the standard normal distribution. The results in the graph indicate that as the dimensions increase both the variance and mean of patterns decreases, with variance losing over half of its value from two dimensions to four.

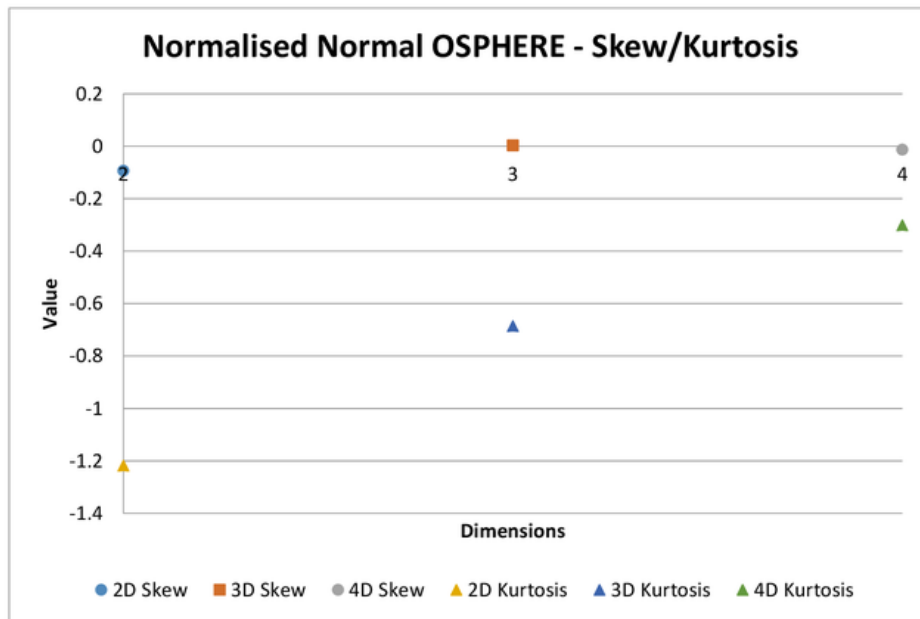


Figure 51: Skew and kurtosis of normalised OSPHERE.

Figure 51 shows the results gathered when normalising the OSPHERE algorithm. The results shows that an increase in dimensions linearly effects the kurtosis of the patterns, going from an average kurtosis of -1.202585 in two dimensions to an average of -0.322894 in four dimensions. The skew of the data is also affected and tends to zero on three dimensions or more.

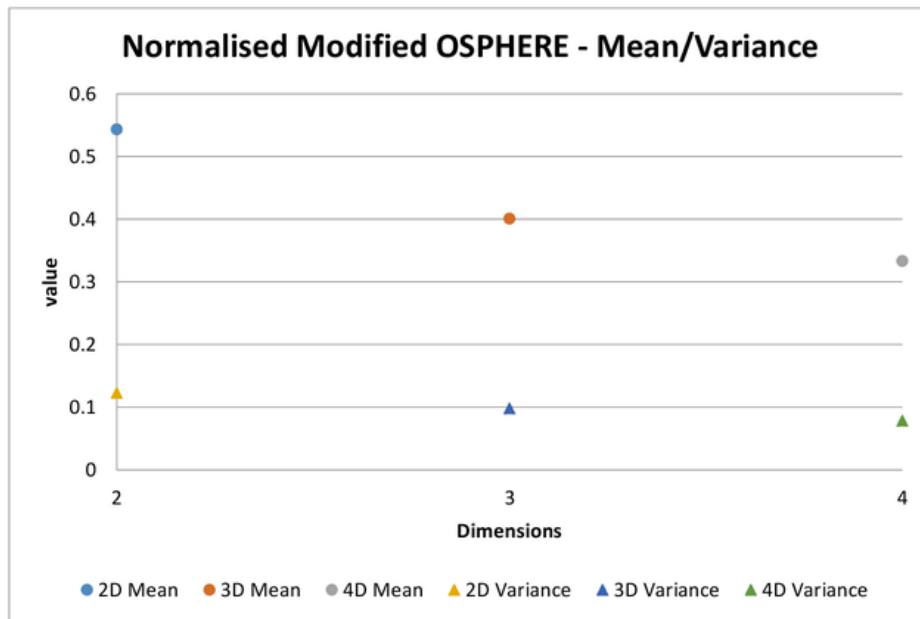


Figure 52: Mean and variance of a modified normalised OSPHERE.

Figure 52 shows the results gathered when running tests on a normalised modified OSPHERE algorithm. Just as previous tests the OSPHERE algorithm has been modified by changing the normal distribution inside of the algorithm to a uniform real distribution to see how distribution is affected. The graph indicates that an increase in dimensions has a limited effect on the variance of the pattern as it averages around 0.1 for all dimensions. However the mean of the patterns linearly decreases as dimensions increase.

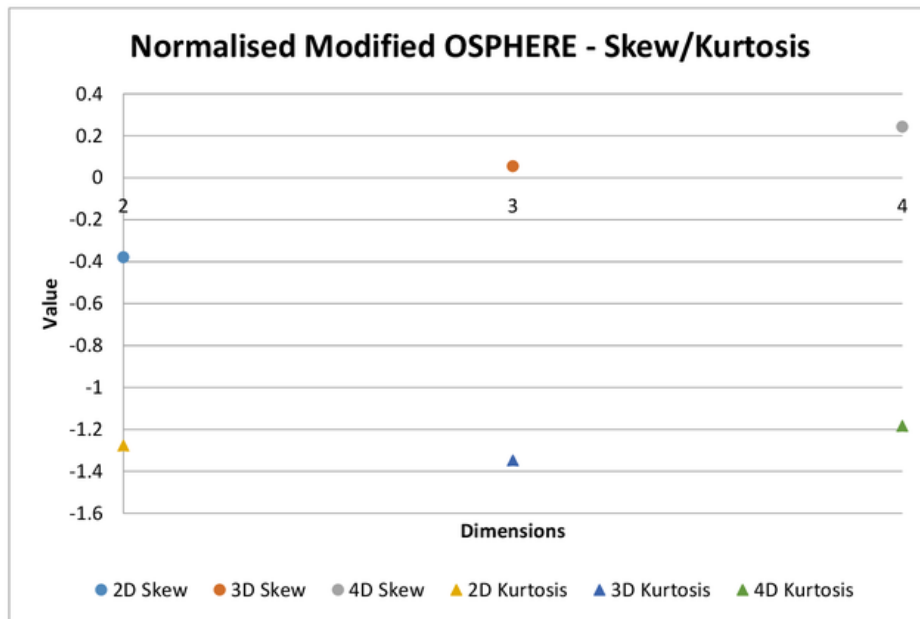


Figure 53: Skew and kurtosis for modified normalised OSPHERE.

Figure 53 shows the skew and kurtosis results of the modified OSPHERE algorithm. The results show that the increase of dimensions has little change on the kurtosis of the patterns as the average kurtosis in two dimensions is -1.310672 and -1.234725 in four dimensions. The skew of patterns however changes drastically with an increase of dimensions.

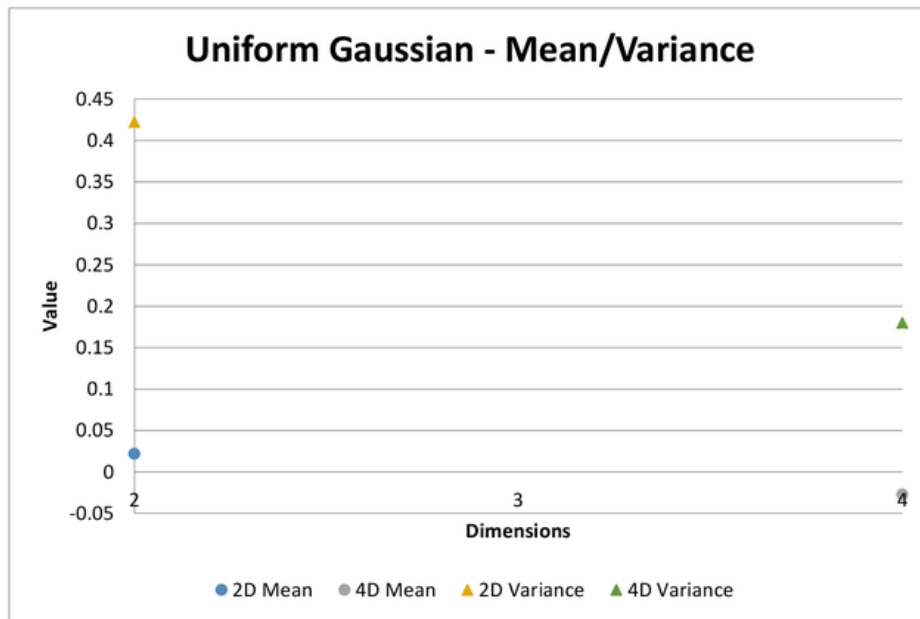


Figure 54: Mean and variance of the Uniform Gaussian distribution.

Figure 54 shows the mean and variance of the uniform Gaussian distribution. This distribution is the normalised implementation of Box-Muller. From the graph it can be seen that as the number of dimensions increases the variance inside of the patterns decreases.

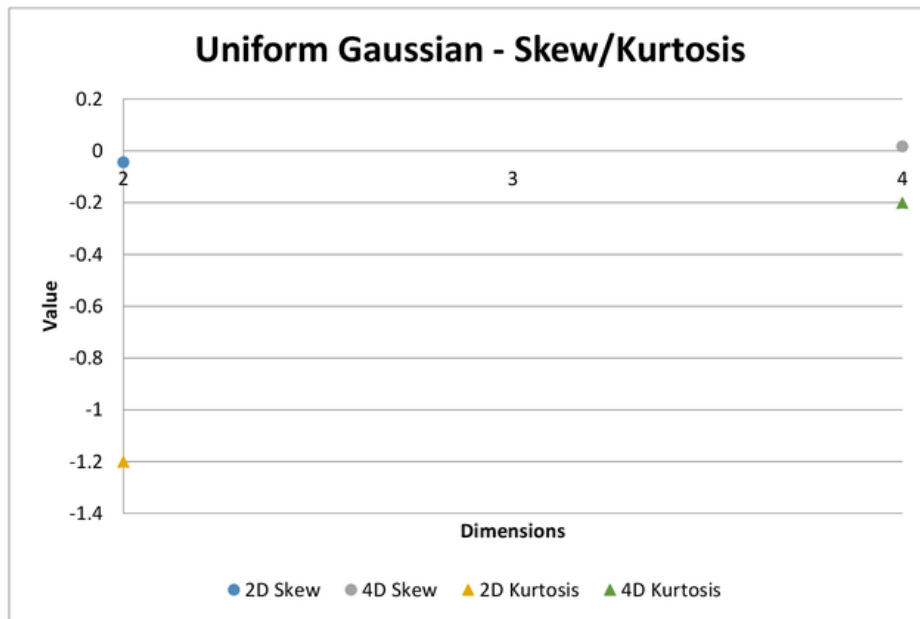


Figure 55: Skew and kurtosis of uniform Gaussian

Figure 55 shows the skew and kurtosis gathered when testing the uniform Gaussian distribution method. The graph indicates that when running the tests when the dimensions were increased from two to four, the kurtosis of the pattern increased significantly.

4.7 Results Discussion

From the gathered results it can be seen that when testing with a non-normalised random distribution that the mean of the patterns will always tend towards zero, however a correlation was discovered, that as the dimensions increased the variance inside of the patterns was decreasing. With a decrease in variance it shows that the distribution of points is beginning to cluster around the mean of the patterns. This can be seen in the figure 41, showing the skew and kurtosis of non-normalised random distribution. This graph shows that the skew of the patterns maintains a presence around zero indicating that the pattern is symmetrical, however the kurtosis as expected in random distribution for one dimension equates to the value -1.2 indicating that in one dimension the distribution is uniform. Once the dimensions increase it can be seen that the kurtosis increases from a uniform distribution of -1.2 to an average of -0.5676327 in four dimensions indicating that the distribution is moving towards a normal distribution with a value of 0 which can be linked to the variance approaching zero, which indicates the probability of patterns being generated around the mean.

As an alternative, running tests again while normalising the random distribution, it can be seen in figure 48, that the mean of the patterns stays around zero just as the non-normalised patterns had. However the variance of the patterns can be seen to increase when moving from one to two dimensions moving from an average of 0.3312277 to 0.4154632 respectively. The addition of more dimensions contributed to the same pattern seen in the non-normalised results, in which by the fourth dimension, the variance of the patterns was the lowest of them all at an average of 0.1869398, again indicating that the distribution pattern is heading towards a normal distribution compared to a uniform, shown in figure 49. From figure 49 it is shown that skew of the patterns while there is variation around zero for one and two dimensions for further dimensions it averages around zero indicating symmetry. The kurtosis of the patterns however maintains uniformity for one and two dimensions but in further dimensions the kurtosis begins to increase towards zero with an average in four dimensions of -0.7181631.

The results for non-normalised and normalised random distribution shown that random distribution in higher dimensional space does not work, as neither a uniform distribution nor normal distribution were achieved in any dimensions higher than two.

When running tests on a non-normalised Fishmans OPSHERE algorithm it can be seen in figure 42 that the mean of the patterns averages close to zero for two to four dimensions. Due to limitations of the OPSHERE algorithm it could not be tested in one dimension as it is

used to generate points on a circle or sphere which is impossible in one dimension. A correlation however can be established from figure 42, in that as the dimensions increase the variance inside of patterns decreases from an average of 0.4100045 in two dimensions to 0.1882222 in four dimensions. This pattern again indicates that the distribution of patterns is moving from uniform towards normal. Evidence of this can be seen in figure 43, in which the kurtosis can be seen moving from an average of -1.2 in two dimensions indicating a uniform distribution to -0.3407223 in four dimensions moving towards a normal distribution. The skew of the pattern stays around zero for each dimension showing symmetry in distribution.

Normalising tests for the OSPHERE algorithm produced similar results to the non-normalised versions specifically for the kurtosis of the patterns, which for two dimensions averaged a kurtosis of -1.2, while in four dimensions averaged -0.312759, producing almost identical results produced by the non-normalised OSPHERE algorithm. The results show that normalising the OSPHERE algorithm had no affect on the distribution as a whole, proving that while the OSPHERE algorithm can produce uniform results in two dimensions, higher dimensions move away from uniformity towards a normal distribution.

Running tests on a modified version of the OSPHERE algorithm in which the normal distribution was changed with a uniform real distribution, showed that in the non-normalised version of the tests, the kurtosis of patterns was uniformly distributed for two to four dimensions as shown in figure 45. However the skew of the patterns can be seen transforming from a positive distribution to a negative distribution as the average skew changed from -0.3447482 in two dimensions to 0.2428984 in four dimensions. Compared to other tests, the modified OSPHERE algorithm has a much higher mean than most with an average of 0.5 for two dimensions lowering to 0.3 in four dimensions', showing the mean is not centred on zero.

Upon normalising tests run with the modified OSPHERE algorithm, patterns can be seen in figure 52, showing that just as the non-normalised version, the mean and variance in both methods are nearly identical. The same can be seen in figure 53 showing the skew and kurtosis of the normalised modified OSPHERE in which the skew and kurtosis of the patterns mirror the results from the non-normalised tests. From the results gathered from testing the modified OSPHERE algorithm it shows that this method of distribution is capable of producing uniform results in four dimensional space.

The final testing showed non-normalised Box-Muller and normalised Box-Muller known as uniform Gaussian. The results for non-normalised Box-Muller as seen in figure 47, show

the skew and kurtosis of the results. From the graph it can be seen that while the skew of the data patterns is low and close to zero, identifying symmetry in distribution, the kurtosis when used in two and four dimensions exceeds the range of zero for a normal distribution with an average kurtosis of 0.5530167 in two dimensions and increasing to an average of 0.9780888 in four dimensions. When running the tests a second time and normalising the results to use the uniform Gaussian distribution it can be seen in figure 54, that while generating points in two dimensions the method was capable of producing a uniform distribution with an average of -1.204053, however the increase into four dimensions sees the kurtosis of the data pattern increase to -0.2763541, moving towards a normal distribution.

Chapter 5 - Evaluation

After running tests it was shown that as expected the random distribution used was not capable of producing a uniform or normal distribution of patterns in any dimension greater than two. The same result show for Fishmans OSPHERE algorithm, as the non-normalised nor normalised versions of the method were capable of producing a uniform result once the dimensions had exceeded two. The reasoning behind this is unknown.

The Box-Muller and its counterpart of uniform Gaussian too did not meet objectives as the non-normalised Box-Muller expected to produce a normal distribution produced a kurtosis of nearly one when used in four dimensions exceeding the value of a normal distribution by one. However when normalised and becoming the uniform Gaussian distribution it produced the expected output in two dimensions, producing a uniform distribution but when used in four dimensions the kurtosis values increased to -0.2 increasing to an almost normal distribution. The reasoning behind this is too unknown.

The only tests to meet the aim of a uniform or normal distribution were the results produced by the non-normalised and normalised modified OSPHERE algorithm in which a uniform distribution was used instead of a normal distribution. These methods both produced results showing that they could distribute patterns in four dimensional space with a uniform distribution and a low skew in data to show symmetry.

If this project was too be repeated again, a different metric in which to measure the sampled data patterns could be researched instead of measuring the kurtosis of the sampled data, as kurtosis may not be an accurate representation of distribution. Also finding an alternative to the GSL package would remove limitations of data processing and analysing.

Chapter 6 – Conclusion

From the work and tests done in this project it has shown that while the researched methods of distribution are capable of producing either uniform or normally distributed patterns in lower dimensional space such as one or two, when moving up into higher dimensions uniformity or normality is lost. One method did however meet the objective of creating a uniform or normal distribution, although it was a modified version of an algorithm which was entirely created for testing purposes.

The conclusion drawn from these results indicates that the researched methods of distribution did not produce the expected results when moved up into four dimensions as many of them started approaching a normal distribution but never reaching a kurtosis value of zero. The project did however meet its aims and objectives of researching and developing methods in multidimensional space with the aim of moving into four dimensions and testing for uniformity and normality.

Future research that would provide beneficial to this project would be researching into the values of kurtosis, as the univariate values are commonly known and are the ones used in this project, although research into the multivariate kurtosis and its resulting values may provide more insight to the distribution in higher dimensions.

Other future work that could be looked at would be looking into the current methods of distribution used and move them into higher dimensions past four. As tests showed that many of the distribution used were approaching a normal distribution in four dimensions. If the trend continues, these methods may produce a normal distribution when used in dimensions higher than four. Other distribution methods could also be looked at, an example of one is shown in [18] in which points are uniformly placed by using a random journey around the sphere.

Chapter 7 - Reflection

My time during this project has highlighted many of my strengths and weaknesses.

Although during times when feeling overwhelmed, I've come to appreciate the amount of time, dedication and planning that must go into a project of this size and in doing so overcame the feeling of fear and doubt. I've also come to realise that with plenty of research and time to absorb the information available to me, that not all tasks are as difficult as they first appear to be. During the development of this project I noticed an area of work in which I need to improve on and that is during the design of my programs. This project has taught me that a design that seems logical and reasonable in my head is not always the most functional design and more often than not leads to very complicated and cluttered designs. If I was to undertake a project of this scale again, increasing the functionality and simplicity of my designs would be an area I'd spend time to improve on. As well as focusing more on time management, as I always underestimated the amount of time I'd need in order to complete certain sections of the project, often feeling rushed.

Chapter 8 – Bibliography

- [1] G. L. Jenkins and M. E. Dacey, "A Unit Quaternion Based SOM for Anatomical Joint Constraint Modelling," 2014.
- [2] K. Mehrotra, C. K. Mohan, and S. Ranka, "Introduction", *Elements of Artificial Neural Networks*. Cambridge, MA, USA: MIT Press, 1997.
- [3] R. Rae and H. J. Ritter, "Recognition of human head orientation based on artificial neural networks," *Neural Networks, IEEE Trans.*, vol. 9, no. 2, pp. 257–265, Mar. 1998.
- [4] S. Winkler, G. Hirzinger, P. Wunsch, M. Sc, P. Dr, W. Mecklenbrucker, and T. U. Wien, "Model-Based Pose Estimation of 3-D Objects from Camera Images Using Neural Networks." 1996.
- [5] N. I. Badler, J. O'Rourke, and B. Kaufman, "Special problems in human movement simulation," *ACM SIGGRAPH Comput. Graph.*, vol. 14, no. 3, pp. 189–197, Jul. 1980.
- [6] A. Watt and M. Watt, *Advanced Animation and Rendering Techniques*. New York, NY, USA: ACM, 1991.
- [7] P. Baerlocher and R. Boulic, "Parametrization and range of motion of the ball-and-socket joint," *IFIP Adv. Inf. Commun. Technol.*, vol. 68, pp. 180–190, 2001.
- [8] G. L. Jenkins, "Evolved neural network approximation of discontinuous vector fields in unit quaternion space (s 3) for anatomical joint constraint", PhD Thesis , University of Glamorgan, no. August, 2007.
- [9] F. S. Grassia, "Practical Parameterization of Rotations Using the Exponential Map 1 Introduction 2 Evaluation of Common Parameterizations," vol. 3, pp. 1–13, 1998.
- [10] S. Aravindan and P. Kaleeswaran, "Pose estimation of a low altitude aerial vehicle using quaternion theory and kalman filter," *Recent Adv. Sp. Technol. Serv. Clim. Chang. 2010 (RSTS CC-2010)*, no. 2, pp. 217–222, Nov. 2010.
- [11] M. Abrate and F. Pollastri, "Spiral tessellation on the sphere," *IAENG Int. J. Appl. Math.*, vol. 42, no. August, pp. 129–134, 2012.
- [12] D. Eberly, "Tessellation of a Unit Sphere Starting with an Inscribed Convex Triangular Mesh," pp. 1–12, 2008.

- [13] H. S. M. Coxeter, *Regular Polytopes*, "Polygons and Polyhedra", Dover Publications, 1973.
- [14] T. Davis, "Geodesic Domes," pp. 1–13, 2011.
- [15] G. Roger, "Using a Rigid map for quaternion based constraint validation," BSc(Hons) , University of Wales Trinity Saint David, 2014.
- [16] S. R. Bowling, M. T. Khasawneh, S. Kaewkuekool, and B. R. Cho, "A logistic approximation to the cumulative normal distribution," *J. Ind. Eng. Manag.*, vol. 2, no. 1, pp. 114–127, 2009.
- [17] P. Kritzer, H. Niederreiter, F. Pillichshammer, A. Winterhof, C. Aistleitner, J. Beck, M. Drmota, D. Bilyk, J. Dick, H. Faure, and others, *Uniform Distribution and Quasi-Monte Carlo Methods: Discrepancy, Integration and Applications*. Walter De Gruyter Incorporated, 2014.
- [18] D. Kirk, " Uniform Random Rotations", *Graphics Gems* 3. 1992.
- [19] D. E. Knuth, *The Art of Computer Programming, Volume 2 (3rd Ed.): Seminumerical Algorithms*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1997.
- [20] D. Lee, J. D. Villasenor, S. Member, W. Luk, and P. H. W. Leong, "the Box-Muller Method and Its Error Analysis," *October*, vol. 55, no. 6, pp. 659–671, 2006.
- [21] G. S. Fishman, "Generating Samples", "Monte Carlo." 1999.
- [22] L. T. DeCarlo, "On the meaning and use of kurtosis," *Psychol. Methods*, vol. 2, no. 3, pp. 292–307, 1997.
- [23] D. P. Doane and L. E. Seward, "Measuring Skewness : A Forgotten Statistic?," *J. Stat. Educ.*, vol. 19, no. 2, pp. 1–18, 2011.
- [24] R. O. Process, "Rapid Object-Oriented Process for Embedded Systems," *Real-Time Syst.*, pp. 1–40, 1999.