University of Glasgow | School of Computing Science

Honours Individual Project Dissertation

Developing and Evaluating Extensions to the 3D Cross Section of 4D Geometry

Joe Subbiani

March 18, 2022

Abstract

Four dimensional space is a mathematical concept that cannot be visualised in its entirety by humans. Several paradigms exist to visualise four dimensional objects in 3D, but a viewer will only be able to see a fraction of a 4D object at one time.

Ray marching over signed distance functions provide a quick method of rendering 3D cross sections of a wide range of primitive 4D shapes. 3D cross sections of 4D objects provides a platform to develop extended visualisations to provide a user with more information about a 4D object. Such extensions have been explored in other literature. This paper aims to evaluate the effectiveness of a series of extensions to 3D cross sections of 4D objects; each representation providing the user with more information in order to help them interpret and manipulate the object they are handling.

To properly evaluate each extension of the 3D cross section, a system to rotate objects in $\mathbb{R}^4$ without gimbal lock was employed. With the use of the rotor, an entity from the geometric algebra, participants of the experiment were capable of rotating 4D objects in all six degrees of rotational freedom.

Following the experiments, it was found that the limited time of an experiment was likely too harsh. Given more time, an experiment could be run allowing participants to build a more sturdy foundation in understand 4D geometry, allowing them to properly explore and evaluate each proposed extension to the 3D cross section.

From the data, it can be concluded that extensions to the 3D cross section that display less information on screen are preferred by novice users. representations of geometry with higher overheads tend to overwhelm users with limited experience with geometry in $\mathbb{R}^4$. An abstraction of 4D rotation to 3D rotation, serves as a good tool for teaching users to differentiate the different planes of 4D rotation, however it has very limited use outside of this. Viewing objects from other angles proved to mislead users, as without proper training, they tended to incorrectly identify patterns in the geometry when under continuous rotation. Viewing several cross sections of the same object simultaneously was a preferred extension by participants, but there is not enough good data to conclude as to whether it actually benefited a users understanding of 4D geometry.

Acknowledgements

I would like to thank my supervisor, Dr. John Williamson for the constant support and advice throughout the project, within and outside of our weekly meetings. Thanks to his guidance and support, I was able to take this project as far as I have, and fully explore the complexities of the fourth dimension.

Education Use Consent

Contents

1 Introduction

1.1 The Fourth Dimension

The world is described as three dimensional Euclidean space, conventionally referred to as $\mathbb{R}^3$. Three axes, conventionally labeled x, y and z, define the three degrees of translational freedom an entity can move within. All three axes that make up Euclidean space exist perpendicular to each other. Four dimensional space, conventionally referred to as $\mathbb{R}^4$, is the mathematical concept where a fourth perpendicular axes, conventionally labeled w, exists perpendicular to all of the other three dimensional axes. It is impossible to visualise four dimensions in its entirety and often must be abstracted to an analogues 3D to 2D example to reason and understand the behaviour of four dimensional entities.

1.2 Opportunities for Exploration

Handling four dimensional space presents a number of opportunities and challenges that come with the desire to explore, interpret and manipulate higher dimensional objects. The most immediate challenge is that of being able to manipulate a four dimensional object. Rotating an object in a dimensional space greater than two is a non trivial challenge. The most common method of rotation in 3D cannot be extended to higher dimensions. There are several ways in which users can interact with 3D objects; another non-trivial challenge is finding intuitive ways in which a user can manipulate a 4D object through a two dimensional user interface.

To understand the orientation of an object, a user needs to be able to differentiate similar faces of that object from each other. Colour, patterns and textures can be applied to an object in order to assist in comprehending the current pose of an object in comparison to an otherwise visually similar counterpart. An example of this is a cube rotated 90 degrees ($\frac{\pi}{2}$ radians) and the same cube rotated by 270 degrees ($\frac{3\pi}{2}$ radians). The un-textured cubes would appear to be the same despite not being in the same orientation.

In this paper several methods will be applied in an attempt to assist a users understanding of higher dimensional space. Users will be provided with a series of extensions to the visualisation of four dimensional geometry. Such methods include: viewing the object from other angles; attempts to visualise the object in its entirety spanning along the w axis; and, abstractions of the rotation of an object to help interpret higher dimensional rotations.

1.3 Motivation

Three dimensional space is very neat. There are three degrees of translational freedom expressed by the three axes x, y and z. There are a further three degrees of rotational freedom that are most commonly expressed as rotations about each axis. This supposed tidiness has lead to misconceptions that are so heavily rooted in the average persons understanding of geometry, such that nearly every digital 3D system is built upon a mathematically impure foundation. In the

3D world, these misconceptions do not matter, and the mathematics is still sound. Quaternions, the most common method of controlling or interpreting rotation, is heavily used in gyroscopic devices, interactive digital 3D environments, animation and robotics. More abstract practices such as physics or mathematics, however, often need to consider geometry in higher dimensional spaces.

A variety of fields within science and engineering utilise the visualisation of higher dimensional spaces to tackle a variety of problems. To more intuitively understand a given problem defined in *n* dimensional space it is often suitable to reduce the dimensionality of the problem. However, when a problem is reduced to $\mathbb{R}^3$ or $\mathbb{R}^2$ it may become somewhat trivial. Visualisation of 4D objects can often serve as the "bridge from the 'trivial cases' to the 'nontrivial cases'", (Zhou 1991). Such problems that utilise the visualisation of four dimensional space include: Analysis of differential geometry; collision detection; analysis of 3D objects in motion; and, scalar-fields in 3D space; among others (Zhou 1991).

1.4 Aims

This paper aims to answer questions related to which representation of 4D space, presented as an extension to the 3D cross section, is the most effective in conveying the properties of four dimensional geometry to a user with limited experience. Which representation is the most effective at representing the surface of a 4D object? And which representation is more effective at conveying the rotational pose of an object in $\mathbb{R}^4$? Furthermore, does a users understanding and ability to manipulate 4D objects increase with practice? And can advantages of each representation be realised and digested by potentially inexperienced users?

As per the motivation for this project, an effective tutorial was compiled explaining the foundations of four dimensional geometry. A system to view and interact with 4D objects was developed in Unity, with the capability to showcase a wide variety of 4D shapes using ray marching. Research into intuitive methods of user interaction was explored, and a system to rotate an object without gimbal lock was employed.

This paper proposes several extensions to taking 3D cross section of a four dimensional objects. Each extension is designed to emphasise a different aspect of geometry. The project evaluation explores the effectiveness of each extension in assisting users to interpret higher dimensional spaces, and understand the surface, rotation and pose of a 4D object. Given a series of challenges, a study was conducted that allowed users to manipulate several 4D objects in order to evaluate their understanding of geometry, and compare their understanding against each representation of a 3D cross section. Metrics such as time taken, confidence and accuracy were measured, with each metric being applied to several tasks; with each task focusing on a different aspect of geometry, to evaluate a users understanding of how a cross section of a four dimensional object changes under rotation.

2 Background

2.1 Representations of the Fourth Dimension

There are many ways to represent the fourth dimension. Arguably the most intuitive understanding is that of a 3D cross section. As with how a sphere in $\mathbb{R}^3$ can be considered a stack of infinitely thin circles that vary in size from top to bottom, along the z axis; a four dimensional sphere can be considered as a stack of 3D spheres that vary in size as you move across the w axis.

The 3D cross section is limited given that only a slice of the shape can be seen at any given time. Stereographic projection can improve upon this by allowing a viewer to visualise as much of the shape as their field of view permits. Stereographic projection in $\mathbb{R}^3$ takes a 3D sphere and maps its geometry across a 2D plane. To project another shape, for example a cube, its geometry must be projected beforehand onto the surface of a sphere (Radiya-Dixit 2017). Stereographic projection in $\mathbb{R}^4$ takes a spherical projection of a 4D object, and maps it across a 3D environment. A viewer can stand in the center of the 3D environment and observe the projected geometry. This approach of visualising 4D space is well suited to virtual reality, however it is complex to interpret in a meaningful way, and abstracting an understanding to lower dimensions is nontrivial.

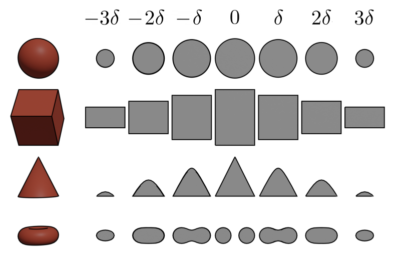
2.1.1 Extensions to the 3D Cross Section

Through simple discussion with peers, a 3D cross section of a four dimensional object appears to be a far more intuitive concept to grasp compared against stereographic projection. The aims of the project are to investigate how one may extend a visualisation of a 4D object to provide a viewer with more information to assist their understanding of higher dimensional space. The 3D cross section of a 4D world appears to be the most appropriate visualisation, given it allows a user to abstract higher dimensional geometry to lower dimensional spaces, to assist in their understanding of higher dimensional geometry.

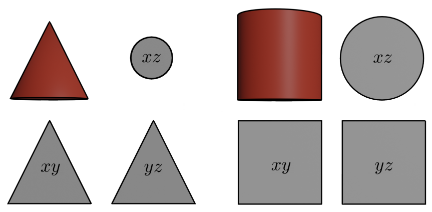
An extension to a 3D cross section involves providing the viewer of the object with more information about the object they are exploring. Several such extensions have already been explored conceptually or even fully developed.

Showing several cross sections beside one another, where each cross section is offset by a multiple of distance δ along the w axis, is a simple means of trying to provide more information to a viewer about the shapes geometry. As shown in the 3D example fig. 2.1a: Several 2D circles taken at regular intervals along the z axis, that circularly increase and decrease in size, provides a viewer with enough information to deduce that the object being displayed is likely a 3D sphere.

Kageyama (2015) proposed such a system, displaying several cross sections of a 4D object along an ovular path, with the goal of providing a viewer with more information about the object. A far greater number of cross sections can be displayed along an ovular path through the use of vertical space, with comparison to a linear path. This allows for a more detailed view of the four dimensional object in its entirety. Kageyama designed each cross section, as they move further away from the central slice, to decrease in size; avoiding potential confusion in conveying the idea that travelling across the fourth dimension wraps around, forming a circular path. Scaling



(a) Several cross sections of 3D objects at δ intervals along the z axis



(b) Cross sections of a 3D cone and cylinder; where each cross section is taken along the specified plane.

Figure 2.1: *Examples of extensions to a 2D cross section of 3D objects.*

each cross section however, can produce some immediate confusion when first viewing an object. For example, taking several cross sections of a 4D cylinder extended along the w axis should produce several 3D spheres of the same size. Kageyama's visualisation, however, will have each of these 3D spheres decrease in size such that a user may interpret the shape as a 3D cross section of a 4D sphere, given it exhibits similar scaling behaviour. If a user were to move across the w axis, they could observe the lack of change in the size of the central 3D cross section and draw the correct conclusion that it is a cylinder. Whilst the ovular display does provide a great deal of extra information to a viewer, it is limited by its obscurity given several objects may have identical cross sections.

Another extension which arguably provides the user with more information about an object, is viewing said object from other angles. As shown in the 3D example fig. 2.1b: take a cone that is extended upwards to a point along the y axis. From the top, a cross section of the object spanning the xz plane will produce a circle. However a cross section along either the xy or yz planes will appear as a triangle.

Polyvision (Matsumoto et al. 2019) is a virtual reality system to view four dimensional objects from multiple angles to quickly and effectively understand the shape being interacted with. Matsumoto et al. (2019) represents 4D objects as wireframes, connecting each vertex of the object by an edge. This is possible when collapsing each vertex of the 4D object to the same three dimensional slice. This is often referred to as the "shadow" of an object. A user may view almost all of an object at once using this technique, in comparison to a single 3D slice; whilst potentially offering a more intuitive understanding, although perhaps less information, when compared to stereographic projection. The wireframe shadow has two main disadvantages. Firstly, curved, smooth objects are typically better represented by their surface, rather than a wireframe. Secondly, complex objects such as the 120-cell, contain hundreds to thousands of edges and vertices, rendering a chaotic image that may be hard to recognise. Viewing the surface of this object likely allows users a better understanding of the geometry. With the goal of teaching others, Matsumoto et al. (2019) restricts translation, rotation and scaling to the third dimension. Rotations in $\mathbb{R}^3$, from another perspective in $\mathbb{R}^4$ may produce a 4D rotation. Whilst the effects of 4D manipulation can be seen in this way, *Polyvision* lacks the ability to directly manipulate an object in 4D space.

2.2 Building a 4D Object

Rendering three dimensional shapes on a 2D screen is an art that has evolved over time and ranges several disciplines. When *Tron* released in 1982, computers were not powerful enough to develop and render triangle meshes in real time; as is the standard for 3D modelling nowadays. Similar films of that era, such as *Dune* (1984) or *Star Wars* (the Original Trilogy) (1977–1983) often used a mix of practical effects and physically modifying film, using techniques such as rotoscoping and double exposure. The team behind *Tron*, however, opted to take advantage of machines to do some heavy lifting. In-spite of the technological limitations the team were able to build a system to combine and render simple but smooth primitive objects. They used a system called Ray Marching (Sheppard 2010).

Today, it is standard to use triangle meshes to build and even render complex models in real time; not just for film, but for real time interactive media such as video games. A mesh consists of vertices, edges and faces. A vertex is a point on the surface of an object that edges connect to. A face is formed by a closed loop of edges. Generally, a face will be made up of three vertices and three edges, creating the most simple 2D shape - a triangle - which form a part of the surface of a 3D mesh.

2.2.1 Mesh Based Rendering

Both Bosch (2020) and Tianli (2018b) have developed real time interactive systems which display four dimensional objects using mesh based rendering. As illustrated by Tianli (2018b), 4D shapes are made up of cells, similar to how a 3D mesh is made up of faces. As with a face in a 3D mesh being constructed of triangles (the most basic 2D shape), a 4D mesh is constructed of the most basic cell, a tetrahedron. A tetrahedron is made up of four triangles defined by faces, vertices and edges. The pentachoron, also known as a hyper-tetrahedron or a 5-cell, is the most primitive 4D object (besides potentially the hyper-sphere). The mesh of a pentachoron consists of 5 tetrahedral sub-meshes (cells). A sub-mesh in this instance refers to a 3D mesh used to build a 4D object, in the same way a 2D faces can build a 3D shape.

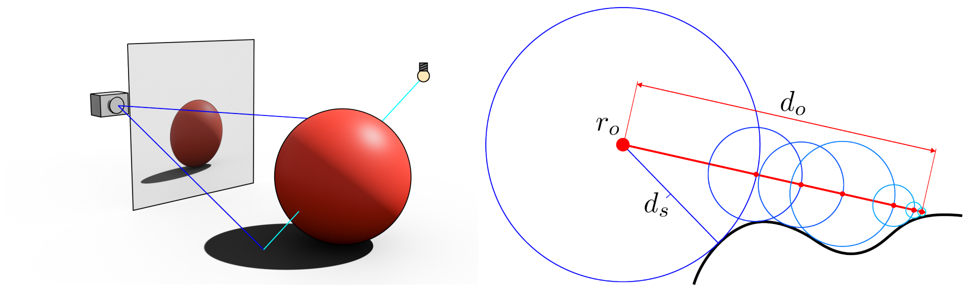
Mesh based rendering allows the creation of very complex higher dimensional objects. Bosch (2020) built a novel game engine implementing four dimensional rigid body dynamics, and several complex shapes, for example, a 4D klein bottle. On the other hand, Tianli (2018b) used the readily available Unity game engine to construct a hyper-cube and hyper-tetrahedron, demonstrating the flexibility of existing software intended for two and three dimensions.

There are two main drawbacks to mesh based rendering: The time taken to construct 4D objects, and the complexity of translating four dimensional geometry to 3D. Simple 4D Platonic solids (Parker 2014) such as the hyper-tetrahedron (5-cell), hyper-cube (8-cell) and hyper-octahedron (16-cell) are fairly straightforward. Developing complex shapes such as a the hyper-dodecahedron (120-cell) or the hyper-iscosahedron (600-cell) becomes a complex and time consuming task. This complexity only increases when trying to develop curve faced shapes such as a hyper-sphere, torus or cone, which are smooth. Smooth geometry generally requires hundreds of vertices as well as smooth surface shading that only add to computational load and development time.

In order to render a three dimensional slice of a four dimensional object, an algorithm must be employed such that a point along an edge that connects two vertices is found and connected, by edges, to all other points that sit along edges of the same object, all lying on the same four dimensional hyperplane. This is a fairly complex task, and only gets more complex when calculating what faces of the slice of the object to display, to render a cohesive three dimensional slice. Tianli (2018a) demonstrates how to find the cross section of an object and the problems associated with it.

2.2.2 Ray Marching Shaders

Ray march based rendering was initially developed due to limitations in real time rendering, as a way to combine easy to define primitive objects into more complex shapes. Despite advancements in rendering capabilities, allowing even low-end computers to render complex mesh based



(a) Ray marching to produce an image *(b) Ray marching algorithm*

Figure 2.2: Ray marching. (a) shows how each pixel of an image is coloured by casting a ray from the camera in the direction of the pixel and reading the surface of any defined geometry. (b) shows the process of a single ray marching towards a surface, given a an origin point and a direction as described in algorithm 1.

```
Data:
  $r_o$ the origin of a ray.
  $r_d$ the direction of a ray.
  $MAX\_STEPS$ the maximum number of iterations.
  $MAX\_DIST$ the maximum distance from the ray origin.
  $SURF\_DIST$ the smallest distance from a surface before deciding not to step any further.
  $SDF(p)$ given a point $p$ along the ray, return the distance to the nearest surface.
Result:
  $d_o$ The distance from the ray origin along the ray path.
begin
    $d_o \longleftarrow 0$
    $d_s \longleftarrow \infty$
    for $i < MAX\_STEPS$ do
        $p \longleftarrow r_o + d_o \cdot r_d$
        $d_s \longleftarrow SDF(p)$
        $d_o \longleftarrow d_s + d_o$
        if $d_s < SURF\_DIST$ or $d_o > MAX\_DIST$ then
            break
        end
    end
end
```

Algorithm 1: The ray marching algorithm

geometry in real time, ray marching has not lost its place in the world. Whilst ray marching has often thought to have evolved into ray tracing, the foundation of photorealistic rendering, signed distance fields (SDFs), the core of ray marching, are still heavily used in volumetric geometry, meta-ball geometry, collision detection and ray marched shaders. An SDF is a function that describes a signed distance to a surface relative to a point in space. Quilez lists several SDFs for primitive objects as well as functions that can be applied to them.

Ray marching (algorithm 1, fig. 2.2) is the processes of calculating the colour of each pixel on a screen by casting a ray from an origin point in the direction of a pixel. The pixel is coloured based on the distance to a mathematically defined geometric surface. Given an origin point and a direction, the algorithm takes the distance to the nearest surface. It then marches forward by that distance to map out the surface detail. (The_Art_of_Code 2018).

Whilst less intuitive than mesh based rendering, there are active communities striving to build interesting and complex geometry with ray marching. *Shadertoy* (Quilez and Jeremias Vila 2013) is a web based platform allowing anyone to create and experiment with shader based geometry.

A significant advantage over mesh based rendering is the ability to render a 3D cross section of a 4D object with minimal effort. The SDF of a 4D object taken over three dimensional space will yield the cross section of the object in a given hyper plane along the w axis. It is not novel to ray march over primitive 4D objects such as the hyper cube or hyper sphere. Often, primitive SDFs taken over $\mathbb{R}^3$ can be easily extended to $\mathbb{R}^4$; and this is the case for many of the SDFs developed by Quilez. Other 4D geometry on the other hand needed to be derived from scratch.

2.3 Rotating a 4D Object

In $\mathbb{R}^3$ there are three degrees of rotational freedom. As there are three axes, the method of rotating an object has commonly been considered as rotating about an axis. In $\mathbb{R}^4$ there are six degrees of rotational freedom, despite there only being four axes. Instead, rotation is considered about a plane formed by two axes. The six planes of rotation in $\mathbb{R}^4$ are xy, xz, yz, xw, yw and zw.

2.3.1 The Problems with Rotation

Rotating an object in $\mathbb{R}^2$ is somewhat trivial. Given a vector v_{xy} rotating an angle θ about an origin point within the xy plane, the vector will follow a circular path dictated by the following 2D rotation matrix:

$$v'_{xy} = v_{xy} \begin{bmatrix} cos\theta & -sin\theta \\ sin\theta & cos\theta \end{bmatrix}, \tag{2.1}$$

Rotating an object in a space greater than two dimensions immediately becomes a nontrivial problem due to its non-commutativity. In higher dimensions, rotation matrices can be composed into a homogeneous rotation matrix. Multiplying a vector with a such a matrix will rotate a vector, but doing so will likely encounter gimbal lock. Gimbal lock occurs when considering each plane of rotation independent to one another. As a result, if a particular plane becomes parallel to another, a degree of rotation is lost and the vector will not rotate as expected.

Introducing the quaternion: an extension of the complex number system. A quaternion has 4 components: x, y and z which describe the axis of rotation, and w which describes the amount of rotation. Quaternions do not suffer from gimbal lock and can be applied to each other to perform several rotations in series.

Quaternions have a problem: they consider rotations as a rotation about an axis. As discussed by Bosch (b), axial rotations are not appropriate as a generalised method of rotation across dimensions. For example, you would not consider a 2D object rotating within two dimensions as rotating about the z axis. It makes far more sense to stay within $\mathbb{R}^2$ and rotate about the xy plane.

2.3.2 Geometric Algebra and The Rotor

Geometric Algebra (or Clifford Algebra) is a field of mathematics describing vector space. Vector space is populated by multivectors, graded by their vector components. Multivectors are defined by their associative, distributive and anti-commutative properties (Baker a). A grade 0 multivector is a scalar number. A grade 1 multivector is just a vector. Grade 2 multivectors are known as bivectors, which build the foundation for the rotor. A trivector is a grade 3 multivector forming a parallelepiped from three vectors. Finally in $\mathbb{R}^4$ a grade 4 multivector is known as a pseudo-scalar.

A bivector b_{ab} is made up of two vectors a and b. Its two vector components define its orientation in space. The bivector is also defined by two important properties: The direction of rotation, from a to b; and, the area of the parallelogram formed by its two vector components as illustrated by slehar (2014). A bivector in the reverse direction, i.e from b to a fulfills the anti-commutative property such that $b_{ba} = -b_{ab}$.

As mentioned above, a rotation should be considered as a rotation about a plane. A bivector defines the plane for a vector to rotate about. As demonstrated by Mathoma (2017), a rotor can rotate a vector following the principle that a double reflection forms a rotation (Mathoma 2016).

In a given vector space greater than $\mathbb{R}^2$, a multivector greater than grade 0 can be projected onto the vector space basis blades. The basis blades of a grade 2 multivector are the planes formed by two perpendicular axes. The xy blade is often referred to as $e_x \wedge e_y$ or e_{xy} or sometimes e_{01}. Therefore a bivector in $\mathbb{R}^3$ can be decomposed into three projections onto the e_{xy}, e_{xz} and e_{yz} planes. A bivector in $\mathbb{R}^4$ can be decomposed into six elements. The outer (wedge) product of two vectors is used to calculate the area of a bivector, and is used to calculate the area of its projections.

It is possible to multiply multivectors using the geometric product; a combination of the inner and outer product. Multiplying a vector v by a rotor R and it's reverse $R^\dagger$ will rotate vector about the bivector component of the rotor. Order matters here.

$$v' = RvR^\dagger, \tag{2.2}$$

Rotors can also be multiplied using the geometric product. This will produce a new rotor. Therefore two rotors; R_a and R_b, can be composed as shown in eq. (2.3) to rotate a vector v and produce a rotated vector v'.

$$v' = R_a R_b v R_a^\dagger R_b^\dagger = (R_a R_b) v (R_a R_b)^\dagger = R_{ab} v R_{ab}^\dagger \tag{2.3}$$

The angle of rotation is dictated by a scalar component of the rotor, similar to the w component of a quaternion. As shown by (Bosch a), the quaternion and the rotor are very similar in concept. The rotor, unlike the quaternion, has the advantage of not leaving the dimensionality of the vector space to rotate a vector, allowing it to be used in n dimensions.

2.4 Interaction and Direct Manipulation

2.4.1 Methods of Interaction in 3D

Common Approaches

Intuitive manipulation of virtual objects in three dimensions is challenging. Excluding mixed reality, generally we can only interact with geometry through a 2D interface. One common method of rotating a 3D object through a 2D screen is with the use of swiping. Swipe-based rotations are very common on touch screen devices, utilising multi-touch as a secondary input to switch dimensions. Assuming the z axis protrudes from the screen, swiping left or right will rotate about the xz plane, up and down will rotate about the yz plane. Two fingers rotating clockwise or anti-clockwise will perform a rotation about the xy plane. Swipe-based rotation is limited to performing rotations only relative to the global axes, opposed to the local axes of a rotated object on screen. As a result, fine tuned manipulation is somewhat challenging, hence it is not common in professional 3D development software.

Alternatively, the most common method of rotating an object takes rotations relative to a local axis. This local axis can be assigned to world space, or rotate objects relative to their real time pose. The method of rotation, despite its popularity, appears unnamed. It will be referred to as a grab ball. The grab ball is made up of 3 arcs. An arc can be selected, restricting the plane of rotation parallel to the arc. The user may drag the mouse to rotate the object, and grab ball, along this plane of rotation. As the grab ball rotates with the object, the user is in control of the objects rotation relative to its local axis at all times. The grab ball provides an efficient method of rotation, but is implemented in different ways throughout various software. For example, *Blender* allows the user to rotate whilst moving the cursor along the path of the arc. *Unity* however requires the user to move the mouse along the path tangent to the arc.

Novel Hardware-based Approaches

Whilst this projects implementation for manipulating geometry is software focused, it is worth noting the research into hardware based solutions. The *Rockin'Mouse* (Balakrishnan et al. 1997) is an example of a hardware centered approach in assisting users with 3D interaction. A regular mouse allows a user to control two translational degrees of freedom (DoF). As a result, a method of switching between dimensions is required in order to fully manipulate geometry in 3D. The *Rockin'Mouse* is a four DoF input device that can be tilted in two DoF. Being in control of four DoF simultaneously, allows users to manipulate geometry in all three dimensions without the need for a secondary input. Results indicate that this hardware allows users to manipulate geometry in 3D 30% faster than with the use of a traditional mouse. The main drawbacks of hardware focused methods of interaction is the expense and lack of accessibility.

Novel Software-based Approaches

Given an object encased in a sphere, a user may roll the sphere to rotate the object inside. The virtual sphere (Chen et al. 1988) allows users to roll the object by dragging their mouse across the surface of the surrounding sphere. dragging the mouse outside of the sphere will rotate the object about the xy plane; the screen the object is being viewed through. This method is similar to swiping, mentioned earlier, but enables the user to control all three degrees of rotational freedom without the need for a secondary input to switch between planes of rotation.

Hysteresis, or path dependance, is when the resulting pose when dragging a mouse from point A to point B may be different depending on the path taken. The virtual sphere exhibits hysteresis, and as such a rotation cannot be undone by reversing the previous action. Arcball (Shoemake 1994) is similar to the virtual sphere, in that the cursor of a mouse is projected onto and dragged across an on-screen sphere. The major difference between the virtual sphere and arcball, is that arcball does not exhibit hysteresis. This is accomplished through two properties: The system utilises the double coverage property of quaternions, such that dragging from either side of the sphere, 180° apart, will complete a full 360° rotation. Arcball consistently compares the initial point A to the updated mouse position B in a single drag, producing one arc of rotation. Combined with the double coverage, any path from A to B will result in the same rotation.

According to Hinckley et al. (1997), there is no evidence that the arcball may be better than the virtual sphere. However, users of the two mechanics generally report the arcball to be more responsive and allow for more control. On the other hand, many users enjoy the virtual spheres slower movement as it emulates the feeling of "pushing" the object around.

2.4.2 Methods of Interaction in 4D

Direct manipulation of 4D geometry is immediately a more complex problem given double the number of planes of rotation. Shoemake (1994) remarks that a pair of arcballs can be used to rotate an object in four dimensions. (Hanson 1992) also proposes a similar approach, taking advantage of a virtual sphere to rotate in 4D.

A primitive solution proposed by Kageyama (2005) explores the idea of using keyboard inputs to rotate an object in all 6 DoF. This system allows users to compose rotation matrices for incremental control over how an object rotates. As the system uses rotation matrices, it is at risk of gimbal lock. Key presses are also discrete, limiting a users precision. A more desireable approach may be to utilise a mouse for a more continuous spectrum of rotation, with an optional incremental feature, as is standard with most 3D software.

Yan et al. (2012) takes advantage of multi-touch screen devices, and the different gestures that can be accomplished with two or more fingers. Multi-touch presents itself as an easier and more intuitive secondary input opposed to the use of a keyboard, where different gestures refer to different rotations.

2.5 Teaching and Evaluation

To evaluate the effectiveness of each extension to a 3D cross section of a 4D object, a participant should have an understanding of what four dimensional geometry is, and how it behaves. For this investigation, it was a priority to ensure this complex mathematical topic can be well communicated to individuals with little-to-no experience of higher dimensional space, prior to conducting the experiment. *The van Hiele Model of Geometric Thinking*, (Šafránková 2012), theorises that their are five levels of understanding geometry, where a student will progress from one level to the next: Visualisation; Analysis; Abstraction; Deduction; and, Rigor, numbered 0 to 4 respectively. Level 0 focuses on the identification and recognition of geometry using simple language. The properties that define a geometry need not be identified. Level 1: students begin basic analysis and identification of geometry, without a need for proofs. Level 2: students create meaningful definitions and are able to justify and reason their understanding of geometry. Level 3: students are able to provide deductive geometry proofs, showing their understanding of formal definitions and theorems. Level 4: students have an understanding of several proofs and can comprehend Euclidean and non-Euclidean geometry.

Šafránková (2012) outlines the five stages of the learning process to comprehensively cover the material and reaffirm a students understanding, as outlined in the van Hiele model. *Information inquiry* is where students receive the material and discover the structure of the information they are handling. At this stage a teacher should use familiar language to help a student relate what they already understand to the new material. *Guided or directed orientation* deals with the exploration of relationships within the material. A teacher may suggest activities enabling students to recognise the properties of a new concept. Discussion is encouraged between the student and the teacher to assist in discovering relationships between properties of the material. *Explanation or Explication*: Students formulate what they have discovered. New terminology is introduced whilst the student shares their thoughts on the relationships discovered. Here a teacher can ensure the correct terminology is being used. The *Free orientation* stage allows for students to solve more complex tasks independently to master the network of relationships within the content. A student may develop their understanding of the properties of geometry as well as develop their understanding of the material in a variety of scenarios. Finally, *Integration* involves the student summarising what they have learned. No new material is presented in this stage.

For the purposes of this experiment, only levels 0 to 2 of *The van Hiele Model of Geometric Thinking* will be explored. A rule of the model is that a student may not progress to level N without having first tackled level $N - 1$. In order to best evaluate the effectiveness of an extension to the 3D cross section of a 4D object, a participant of the investigation should understand the geometry they are working with. Unfortunately, given the limited time to conduct the investigation, and that of an individual experiment, participants would not be able to sufficiently progress through each level, whilst also covering each of the five stages of the learning process. To provide a participant with the best possible foundations for understanding four dimensions, tutorials and question and answer sessions before and after each stage in the experiment were conducted. The experiment was designed around the structure of the five stage learning process as is expanded upon in the experimental design section (section 4.2) of this paper.

3 Analysis and Requirements

3.1 Problem Specification

The core of this project focuses on the development of a system capable of rendering several different types of four dimensional objects, and be able to view these objects through a series of extensions to the 3D cross section. These extensions will be evaluated for their effectiveness in teaching potentially inexperienced users about complex geometry.

For a successful investigation, a focus was placed on the visualisation of 4D geometry, as well as how users learn and interpret this geometry. It is important that the experiment was to be clear and concise. Both quantitative and qualitative metrics were obtained. To properly evaluate each representation of an objects cross section, the experiment required tests probing a participant on the surface, behaviour under rotation and identification of the rotational pose of an object in $\mathbb{R}^4$.

3.2 Limitations

The scope of the project was limited by two main factors: The time that can be spent on development; and, the quantity to quality ratio of experimentation. The project is primarily separated into three components: Research; development; and, experimentation. The development phase took the most time with so many areas for exploration. Such areas include the creation, stable rotation, and manipulation of 4D geometry. To evaluate the effectiveness of the representations that need to be developed however, plenty of time needed to be put aside to plan and run appropriate experiments. Experimentation is, more often than not, limited by the number of participants that choose to be involved. To encourage the greatest number of potential participants to enroll in the experiment, the time commitment and effort cannot be so great that it deters them. Alongside the conscious effort to sign up, the experiment cannot run for such a period of time that it may effect a participants performance. As such, the amount of data that can be collected in one sitting was to be maximised within a limited time frame.

3.3 The Framework

The project was developed in the Unity game engine using ray marching shaders. Unity is readily available to anyone for free; and allows for quick and simple application deployment for web based hosting using WebGL and HTML5. Given the COVID-19 pandemic at the time of the experiment, online hosting was an important consideration to allow for a more accessible experiment in a world where people were having to self-isolate. Not only does this allow for a participants safety, but the potential size of the audience can be increased. Online hosting reduces the commitment a participant has to make, making the experiment more appealing. Time to commute and costs of travel that potential participants would otherwise have to make, now no longer factor into the considerations made by a potential participant when signing up for the experiment.

Ray marching was used rather than mesh based rendering given its comparative ease. Several 4D shapes can be written fairly quickly using ray marching. Although complex geometry with defined faces, such as the 120-cell is a struggle to write using ray marching, render curved geometry using mesh based rendering requires the development of a smooth shading rendering effect. Moreover, the experiment only intended to test users with 4D primitive objects given they are easier to understand and relate to three dimensions. Furthermore, implementing a rendering of the 3D cross section of a 4D object is comparatively trivial using ray marching. To render the three dimensional cross section of a 4D mesh is yet another system to develop that would require more time and research. Generally, ray marching is more suitable to this project in exploring primitive geometry in $\mathbb{R}^4$, and allows for more time to be spent on the research and development of stable rotation and manipulation of 4D geometry, and the creation of a cohesive investigation.

3.4 Prioritisation of Requirements Using MoSCoW

MoSCoW is a prioritisation technique used in project planning. The purpose of MoSCoW is to separate the requirements of a project into four levels of priority: Must have, Should have, Could have and Won't have this time. This process may be revisited several times during a project's development lifecycle; different requirements are often re-prioritised based on project progress or a change in direction.

Must Have

The following requirements are necessary in order to conduct a successful evaluation and were compiled and refined throughout the project's development:

A system must be developed that is capable of rendering 4D shapes. As such, several 4D shapes must be created. In order to test a persons understanding, of the 4D shapes that are to be developed, they must not all be immediately distinguishable from one another. For example, a hyper-cube and a hyper-sphere will never look similar to each other, whereas, the 3D cross section of a 4D hyper-cone in a particular pose will look identical to the 3D cross section of a 4D hyper-sphere.

The aims of the project focus on the development of several extensions to the 3D cross section to find which extensions are better for conveying the complexities of 4D geometry to potentially novice users. Therefore, several extensions to the 3D cross section of four dimensional space should be developed. The extensions to the 3D cross section may take inspiration from the literature mentioned earlier, and attempt to improve upon or emphasise the components of their work that are relevant to this investigation.

To distinguish the faces of shapes, the objects will need to be coloured such that given two objects in a different pose, both objects can be distinguished from one another even if their surface shape appears identical.

In order to prepare participants for the experiment, an introductory tutorial video will need to be assembled to teach and explain the complexities and features of the fourth dimension. This tutorial video must cover the behaviour of geometry in $\mathbb{R}^4$, with appropriate references to the similarities when reducing three dimensional space to a two dimensional cross section. Moreover, the video must explain rotation in four dimensions, as well as an explanation for the behaviour that occurs when rotating an object outside of the dimensions it is being viewed within. The video must have relevant diagrams and animations to appropriately depict the behaviour of 4D shapes in such a way that is related to the following experiment.

Before running the experiments, the tasks must be completed and evaluated several times by myself, and complimented by some preliminary research with a small group of peers to ensure the functionality of the system is as expected, and any restraints such as time limits are within acceptable bounds.

Finally the experiment must have multiple stages of testing to evaluate a persons understanding of the several aspects of geometry. The experiment must not last such a long time that it affects the participants performance. To appropriately evaluate a persons understanding alongside the effectiveness of a representation, qualitative data must be collected from the participant, about their understanding, including open and closed questions.

Should Have

Whilst not a necessity, the following requirements would enhance the investigation, allowing the further exploration of the four dimensional world.

"Free orientation", as stated by Šafránková (2012), is an important aspect in a students learning and understanding of the definition and behaviour of geometry. In order for a participant to master the complexities of geometry in $\mathbb{R}^4$ they should be able to handle 4D objects directly. Evaluating a participants understanding of a shape, its surface and its behaviour under rotation can be conducted without directly interacting with or manipulating an object; however a system should be developed allowing a participant to directly interact with 4D objects in order to perceive, first hand, the effect they have on the cross section of the geometry. For direct manipulation in four dimensions, a 4D Rotor must be implemented to allow for stable interaction, without fear of gimbal lock.

Following the implementation of a system to freely rotate 4D geometry, a challenge should be implemented evaluating a participant's understanding of the geometry through the use of the manipulation of objects on screen.

Could Have

Given the limitations, although it would be nice to include, the following requirements are the lowest priority. The requirements will be explored, but are not necessary to conduct the investigation and will likely not be refined to a satisfactory degree.

It would be desirable to implement a series of tools for manipulation, allowing for intuitive interaction of 4D geometry. Such methods of interaction will be experimented with. The experiments, however, will only implement one method of rotation for the experiment, given the investigation focuses on the ways to enhance the cross section of 4D space, rather than how to manipulate these objects. Therefore, it is of minimal priority to develop several methods of rotation, although during the research and development phase, several methods of interaction will be explored.

4 Design

Given this investigation revolves around a participant's understanding of complex geometry, it is important that they are familiar with the behaviour of this geometry prior to completing the main portion of the experiment: Each extension to the 3D cross section will be evaluated against a series of tasks a user must complete. Prior to this phase, the user will be trained on the complexities of the fourth dimension. Elements will be taken from the *van Hiele model of geometric thinking* as discussed in section 2.5. The different portions of the experiment heavily encouraged discussion with the teacher/observer to ensure a participant has the best understanding of four dimensions possible, prior to and during the experiment.

4.1 Tutorial

A brief discussion was conducted with each participant following their introduction to the experiment and what the experiment would entail. This discussion focused around a participant's prior knowledge with the concepts of 4D. As per the first stage of van Hiele's requirements for teaching, *information inquiry*, students are to receive the material and discover the structure of the geometry presented in this stage. This introduction should use familiar, well-known and understood language. The material received by the students at this stage was compiled in the form of a video tutorial, linked here: `https://www.youtube.com/watch?v=fhnhK7w67_s`

The tutorial was structured such that the world of four dimensions could be built upon piece by piece in hope that students would better digest each component of $\mathbb{R}^4$ in turn without feeling too overwhelmed. To begin with, the concept of a fourth perpendicular axis was introduced, with emphasis that this could not be visualised, and instead a method of viewing 4D geometry would be with the use of a 3D cross section. The 2D cross sections of some primitive 3D shapes, namely the hyper-sphere and a hyper-cube, were showcased. To complement the behaviour of a shape's cross section, the geometry of a hyper-sphere and a hyper-cube were described independent to their cross sections.

The rotation of 4D geometry is a more complex subject. As such, the rotation was broken down into several stages. Rotation in n dimensions must be considered a planar rotation, opposed to axial rotations as most people are used to. Planar rotations are first introduced in 3D and were equated to the axial rotations to assist in a learners understanding. To emphasise how planar rotations work in n dimensions, a 2D cross section of a 3D object is first rotated within the two dimensions of a slice in 3D space. This showcases that an object rotating within a cross section does not change size or shape. On the other hand, a 2D slice of a 3D object, rotating in three dimensions (a rotation involving the z axis) will change size and shape. The parallels are then drawn between the 3D cross section of a 4D object: The 3D cross section will not change size or shape for any rotation within three dimensions, however it will if rotating in four dimensions (a rotation involving the w axis).

4.2 Experiment

An informal discussion took place after the tutorial video to elaborate on the participant's understanding. This discussion would utilise the *explanation or explication* and *integration* phases of the learning process; where a student is asked to formulate what they have learned, and share their opinions on the relationships they have discovered. Here, the teacher may correct terminology. Towards the end of the discussion, the participant would summarise what they have learned ensuring adequate preparation for the coming experiment.

Given the limited time to conduct an experiment, the remainder of the experiment took guidance from the *free orientation* and *guided or directed orientation* stages of van Hiele's learning process, effectively merging them into a single orientation phase. Participants were provided with a 4D hypercube, and a summary of the controls to manipulate the shape. The participant was allowed to spend time observing how the shape may change when undergoing different rotations, as well as how the shape may change when moving it through the hyper plane. Informal tasks were proposed to the participants allowing them to connect relationships of the geometry. An example of such a task included rotating the shape with the use of a 4D rotation such that the colours of two opposing faces of the 3D cross section switch sides of the object. Here participants may learn basic properties of rotations in $\mathbb{R}^4$.

Of the developed extensions to the 3D cross section of four dimensional space, different extensions may have different strengths and weaknesses. Some extensions may excel in describing the surface geometry of an object, whilst others may better convey the rotation of an object in $\mathbb{R}^4$. Therefore the main portion of the experiment was broken down into a series of three tasks, repeated for each representation. Each of the three tasks focuses on a different aspect of geometry: understanding the shape of a 4D object; understanding the rotation of a 4D object; and, the ability to manipulate a 4D object. First, the participant was to be tested on their understanding of the surface of a shape. Given a cross section of a 4D object, the participant should be able to identify the shape on screen. Second, the participant was tested on their understanding of any rotations a 4D object may undergo. Given a 4D object under continuous rotation, the participant should be able to identify the active planes of rotation. Finally, the participant was tested on their ability to manipulate 4D geometry. Given a second object in a random pose, the participant is tasked with matching the primary objects orientation, with that of the secondary object. The randomly orientated object is only randomly oriented along 4D planes of rotation, allowing a user to manipulate both shapes simultaneously in 3D to view all sides of the objects in question.

4.3 Interaction

To intuitively rotate an object in all six degrees of freedom, the method of interaction should be familiar to the user. A common human computer interaction technique for intuitive navigation of a user interface is to have an action made by the user be analogous to a real world action. For example, swiping across the screen to move a digital object aside, is designed to mimic moving a real object in the same way. The same methodology was employed, allowing the user to interact with an object by swiping in the direction of rotation. The 2D screen limits the types of gestures that can be accomplished when manipulating an object in three or four dimensions. Touch screens overcome this limitation through multi-touch. Alternatively, as used in this project, a keyboard used in combination with a mouse can allow for alternative interactions that specify or limit the dimensions being used. To assist a user in rotating an object, a set of x, y and z axes were displayed in the corner of the screen so a user may infer which planes of rotation they may choose to limit. As a result, an intuitive system has been developed, allowing users manipulate objects in any of the six planes of rotation through the 2D user interface. This is expanded upon more in section 5.4.

5 Implementation

To conduct the investigation, four main components had to be developed (fig. 5.1): Building and rendering 4D shapes; rotating and manipulating these shapes; building and running the experiment; and, collecting and analysing the data from the experiments. There would be no project without the ability to render 4D shapes, so this was developed first. The experimental tasks were built up alongside the development of the rotor. The rotor was required to manipulate 4D shapes without gimbal lock. After the experiment was assembled, data analysis tools could be developed independently, as experiments were run.

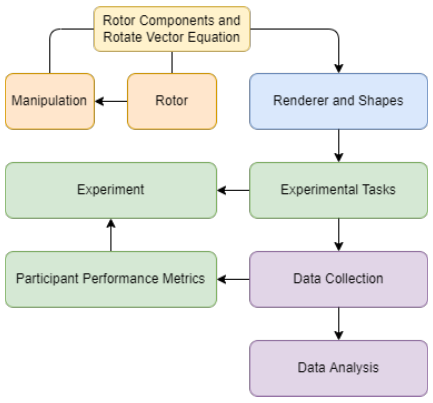


Figure 5.1: *Project Component Diagram*

5.1 Building 4D Objects With Ray Marching

As stated in section 3.3, ray marching shaders in Unity were used to develop and render three dimensional cross sections of four dimensional objects. Ray marching over signed distance functions in Unity is not an uncommon topic within art and game development communities. To render a ray marching shader, a material utilising the ray marched shader must be applied to an object, such as a plane. The scene defined within the shader will be rendered onto the object. Before implementing a system to render four dimensional objects, a foundation was layed in the form of a three dimensional ray marcher in Unity (The_Art_of_Code 2019b). The ray marching algorithm (algorithm 1) casts a ray from an origin pointer towards a pixel on the screen, to determine how to colour this pixel. The algorithm takes a signed distance function (SDF) which returns the distance from a point p along the ray to the surface relative to an origin point. To summarise the basics of SDFs, listing 5.1 shows a signed distance function written in GLSL for a **sphere** in $\mathbb{R}^3$ with radius r. The center of the sphere is offset from the origin. To interpret this, the opposing translation is applied to the point p, along the ray. Using p, the distance to the surface is calculated. Subtracting the radius from the distance to the center of the object defines the surface of the sphere about its center.

```
float sdSphere( float3 p, float r )
{
   p -= float3(1,0,0);
   return length(p)-r;
}
```

Listing 5.1: *The signed distance function for a sphere in* $\mathbb{R}^3$ *with a center at* $(1, 0, 0)$*, and radius* r

The 3D ray marching shader written in Unity defines two vectors describing the ray origin and the hit position of the ray. These vectors are used to define the ray origin and ray direction for each pixel, to calculate what the shader should render. When extending the ray marcher to four dimensions, the majority of the process was enabling the algorithm and its components, such as the SDF, to take 4D vectors (`float4`), rather than a 3D vector (`float3`). Given the components of the shader describing the ray origin and direction exist along the same slice of four dimensional space, only a single 3D slice of the four dimensional space was taken and rendered, giving us the 3D cross section.

With the standard ray marcher implemented, the shader can only interpret whether a ray hits a surface or not. The only information of a 3D shape that can be gathered from this binary value will be its 2D silhouette. In order to render a 3D component of the object, we need light and shadows. Two basic methods of shading include angle dependant light falloff - lambertian diffuse elimination - and shadows cast by other objects obstructing a light source listing 5.2. A surface normal is a vector that is perpendicular to a point on the surface and can be used to measure the angle of the surface relative to another vector. Using the surface normal, lambertian diffuse elimination can be implemented such that a surface's light value will be lower if the difference between the surface normal and the vector to the light source is higher. A shadow cast on to a surface can be taken by ray marching from a surface in the direction of the light source, to check if there is another surface obstructing the path. If there is, the surface's light value can be lowered.

```
float4 GetNormal(float4 p)
{
   float2 e = float2(0.01, 0);

   float4 n = GetDist(p) - float4(
      GetDist(p-e.xyyy),
      GetDist(p-e.yxyy),
      GetDist(p-e.yyxy),
      GetDist(p-e.yyyx)
   );
   return normalize(n);
}

float GetLight(float4 p, float4 lightPos)
{
   //angle dependant fall off
   float4 lv = normalize(lightPos-p);
   float4 n = GetNormal(p);
   float light = clamp(dot(n,lv), 0., 1.);

   //shadow
   float4 so = p + n * SURF_DIST * 2.; //shadow origin
   float4 sd = normalize(lightPos-so); //light direction
   float d = Raymarch(so, sd);
   if( d < length(lightPos-p) ) light *= 0.1;

   return light;
}
```

Listing 5.2: *Given a light position `lightPos`, shade the surface based on the fall off angle and cast a shadow, if a surface is obstructing another surface from the light source*

5.1.1 Derivation of 4D Shapes

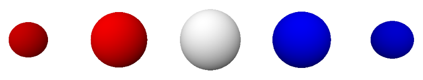
Ray marching excels at creating infinitely complex shapes, such as high dimensional fractals. Participants of the experiment may have little to no experience with higher dimensional geometry. To effectively evaluate each extension to the 3D cross section, the participant needs to understand the information they are seeing on screen. The best way to convey complex ideas is with simple examples: ergo, using geometric primitives, rather than complex geometry.

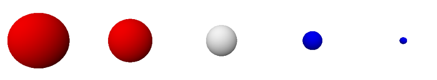
Of the 3D SDFs derived by Quilez and The_Art_of_Code (2019a), the sphere, box, and capsule were fairly trivial to translate directly to four dimensional space. A **4D hyper sphere** takes a radius, with the surface spanning equally along all four axes. A **4D box** takes a 4D vector where each component describes the height, width, depth and length of the box along each of the four axes respectively. Similar to how the sphere subtracts the radius from the distance to the origin point, the box subtracts the 4D vector describing the dimensions of the box. To flatten the faces of the box, without the distance function stepping passed the surface, the vector is combined with the zero vector using a boolean intersection, with the use of `max(a, b)`. The derivation is best described by The_Art_of_Code (2019a). Finally a **4D capsule** can be described as a line connecting two points, A and B. The line is given a thickness via subtracting a radius. Just as with the sphere, the radius spans in all four dimensions.

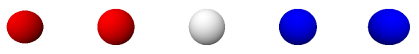
The torus is an interesting shape; two variations were developed for this project. A **4D torus with two radii**, like a 3D torus, can be described by a major radius, r_1 and a minor radius r_2. r_1 is the main ring of the torus, and r_2 is the thickness of this ring - its radius. When you take a 2D slice of a 3D torus it may appear as a pair of circles separated by a distance of $2(R_1 - R_2)$ as shown in fig. 2.1a. Similar to the cross section of a sphere, it makes sense to have the radius r_2 span across the fourth dimension. The 3D cross section of a four dimensional torus therefore may appear as a pair of 3D spheres. Alternatively, consider a **4D torus with three radii**: r_1, r_2 and r_3. Similar to how the 3D torus can be described as a circle with a thickness, a 4D torus with three radii describes a circle defined by radius r_1, with radius r_2 surrounding the circle that is given a thickness defined by the radius r_3. Essentially this produces a circle with a torus sweeping across its path. Therefore, a 3D cross section of this torus may appear as two tori with a major radius of r_2 and minor radius of r_3 separated by a distance of $2(r_1 - (r_2 + r_3))$.

The 3D cross section of a **4D cone** in some instances appears as a 3D cone, and in other instances appears as a 3D sphere, in the same way a 3D cones' 2D cross section may in some cases appear as a triangle or a circle (fig. 2.1b). The cone SDFs developed by Quilez and others define the cone by its height and slope. This approach did not extend well to 4D, and defining an instance of a cone was rather unintuitive. The developed SDF for a cone takes a height and the radius at the base of the cone. This approach extends well to $\mathbb{R}^4$, as rather than a defining the cone as a surface function of angle and height, it defines the slice of the cone at any given cross section.

Of the shapes defined thus far, the 4D box is the only shape with edges, meaning it would obviously stand out against the other curved surfaces. A **4D tetrahedron (pentachoron)** was created to present another platonic solid, showcasing their similarities. The SDF of plane can be defined by its normal vector. For example, The SDF of a plane that spans xy in $\mathbb{R}^3$ would be `p.z`. A tetrahedron is made up of 4 triangular faces, each opposing one of it's four vertices. The SDF for a tetrahedron uses these vertices as the vectors describing the normal of each face. Each of these vectors defines the SDF of a plane parallel to a face of the tetrahedron. By taking the boolean intersection of these planes using `max(a, b)`, the SDF for a tetrahedron is produced. The SDF for a tetrahedron also takes a parameter s, used to scale the tetrahedron. Subtracting s from the distance to the surface, effectively brings the surface further from the center, similar to the radius of a sphere. Constructing a pentachoron is done in a near identical way, just five vertices are considered rather than four.


(a) *Sphere*


(b) *Cone extended along w*


(c) *Capsule extended along w*

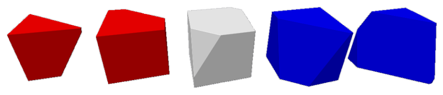
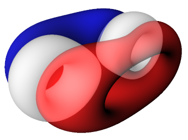
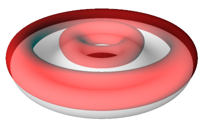
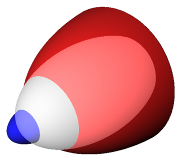

(d) *Rotated Cube*

Figure 5.2: *The* w-axis timeline *extension to the 3D cross section of a 4D object. (a) shows a sphere. Note it scales down as the cross section moves away from the center of the object. (c) shows a capsule – a cylinder with rounded ends – extended along w. There is no scaling between the cross sections. (b) shows a cone extended along w. As the cross section travels from the base to the tip, it decreases in size*


(a) *Torus*


(b) *Torus*


(c) *Cone*

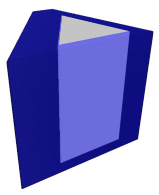
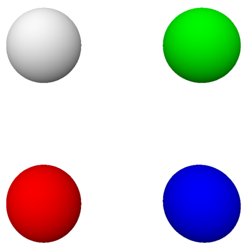
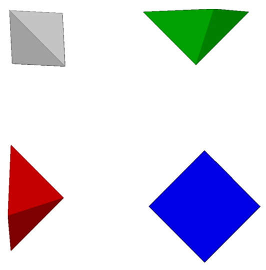

(d) *Cube*

Figure 5.3: *Early development of the* onion skin *extension to the 3D cross section. (a) shows a torus rotated along wx and wy with no shift along the w axis. (b) shows a torus rotated along wy with a small shift along the w axis. (c) shows a torus rotated along wx and wy with a small shift along the w axis. (d) shows a cube rotated along wx and wy with a moderate shift along the w axis.*


(a) *Sphere*


(b) *Pentachoron*

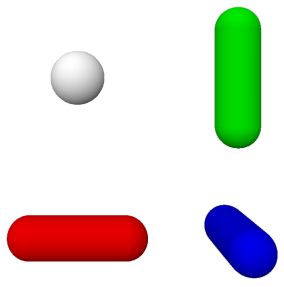

(c) *Capsule*

Figure 5.4: *The* multi-rotational view *extension to the 3D cross section of a 4D object. In each sub figure, the top left shows a regular 3D cross section of a 4D object: S. The bottom left shows S viewed from* 90° *on the xw plane. The top right shows S viewed from* 90° *on the yw plane. The bottom right shows S viewed from* 90° *on the zw plane.*

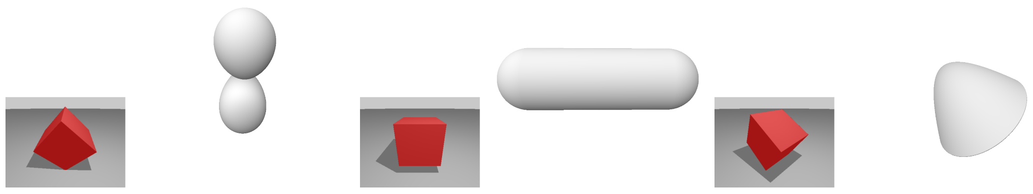

(a) Torus *(b) Capsule* *(c) Cone*

Figure 5.5: *The* 4D to 3D abstraction *extension to the 3D cross section of a 4D object. (a) shows a torus rotated approximately* 45° *about the xw plane, and then* 70° *about the xw plane. (b) shows a capsule initially extended along w, and then rotated* 90° *about the xw plane.*

5.2 Extending 3 Dimensional Cross Sections: Representing 4D Objects

The signed distance function referenced in the ray marching algorithm defines the scene to be rendered. In this function, several other SDFs can be referenced to render several objects within the same scene. To render several SDFs at once, each function may be combined using a boolean union: taking the minimum distance between two or more objects using `min(a, b)`.

Each extension to the 3D cross section took advantage of colouring different shapes within a scene. To colour different shapes within a scene a second function is employed that for the most part is identical to the SDF describing the scene, however it returns a number. Each value the number may take is a material index. To find what material index relates to each object in the scene, a comparison is made between the distance returned by the union of all SDFs in the scene, and the distance to a specific SDF in the scene. If they are equal, then the material index associated with that specific SDF is returned (The_Art_of_Code 2021).

5.2.1 *W-axis Timeline*

The simplest extension to a 3D cross section of 4D space is to render several cross sections at once, each offset by a multiple of the distance δ_1 along the w axis, with each cross section being separated by a distance of δ_2 along the x axis. The core of this concept has been explored by Kageyama (2015). To avoid the aforementioned problems with scaling, each cross section was built to exist side by side one another as shown in fig. 2.1a. Throughout the project, this representation has been referred to as the *w-axis timeline*. Given an object under translation, passing through the hyper-plane, over time, by definition labels each cross section as a glimpse into the past or future, with respect to the primary central cross section.

A drawback of showcasing cross sections side by side is that fewer cross sections can be shown on screen at one time, in comparison to the elliptical design presented by Kageyama (2015). This problem is generally less significant for primitive shapes, such as the ones implemented for this project. The problem becomes more significant when visualising complex shapes with unpredictably changing surfaces. Given this representation provides more information about the geometry of an object over $\mathbb{R}^4$ it is reasonable to expect that it would most effective in shape identification. The *w-axis timeline* may also provide some benefit a user to predictably manipulating a four dimensional object, or identify a rotation more accurately, although this is likely to be not particularly significant, if it all.

5.2.2 *Onion Skin*

Taking inspiration from 2D animation, for its time in development, the following extension was referred to as the *onion skin* representation. In 2D animation, the animator may enable the onion skin tool to see a set number of frames before and after the current frame so that they may see how the animation will progress as they move forward in time. This representation aims for a similar approach, where the user may see backwards and forward along the w axis, rather than time. This approach follows a similar vein of thinking to the *w-axis Timeline*. The differing factor between the two is that the *onion skin* representation overlays two translucent cross sections, forward and backward along w, around the main cross section as shown in fig. 5.3. The goal was to emphasise the geometry of a single object in a more cohesive manner. The transparency of the cross sections were achieved by ray marching over 2 different distance functions. The first SDF contains all three cross sections. The second SDF just contains the central object. Two renders of the scene will be produced. By halving the colour values of both renders and overlaying them above each other, the full colour range is regained and anything that was hidden by one of the translucent objects will now be visible.

The *onion skin* representation was not carried forwards to the experiment phase. Each representation was placed under trial during informal discussions with peers. The participants of this discussion generally had limited experience of the fourth dimension. The purpose of this discussion was to evaluate if a representation had potential before carrying it through to the final experiments. During discussions on objects under rotation, whilst this representation was active, the onion skin effect was described as confusing and hard to follow. Originally the translucent cross sections were set to be a consistent distance along w from the main object, as with the *w-axis Timeline*. It was found that it was not obvious that the central cross section would become either of the other cross sections as it moved along w, as expected. In an attempt to rectify this problem, the translucent cross sections were set exist at fixed intervals along the w axis. Using this, the central cross section could be observed directly to become the translucent cross section when translated in its direction along w. When the central cross section reaches the same position along the w axis as a translucent cross section, the translucent cross sections would be shifted by a fixed amount such that the central cross section would again be directly in between the two translucent cross sections. Despite these efforts, this representation was still deemed to confusing for users as was subsequently set aside for the remainder of development.

5.2.3 *Multi-rotational View*

Polyvision, proposed by Matsumoto et al. (2019) extends a 3D shadow wireframe of a 4D object by introducing the ability to view the object from multiple angles. Viewing an object from multiple angles at once provides a great deal of insight into the geometry, and also in identifying the rotations it may undergo. A similar representation of 4D geometry was applied during this project over 3D cross sections of 4D objects. Throughout the duration investigation this representation has been referred as the *multi-rotational view* or *multi-view* for short. In order to view the same cross section from three other angles, the SDFs for four objects are rendered in total, with the three of the objects rotated 90° about the xw, yw, zw planes respectively. To differentiate the four cross sections they were coloured differently. Conventionally, the x, y and z axes are coloured red, green and blue respectively. Therefore the cross section that was rotated about the xw plane was coloured red, the cross section that was rotated about the yw plane was coloured green and the cross section that was rotated about the zw plane was coloured blue; as shown in fig. 5.4. One of the biggest strengths of the *multi-view* representation is its ability to clearly convey the behaviour of a slice of 4D geometry under rotation; although it can be challenging to comprehend at first. Another positive of the *multi-view* representation is its ability to effectively provide enough information to determine the shape that is being viewed, without requiring a user to first interact with the object to verify their hypothesis.

5.2.4 *4D to 3D Abstraction*

The most risky extension to the 3D cross section of a given 4D object, within this investigation, attempts to abstract 4D rotations (any plane of rotation involving the w axis) to 3D rotation. The goal of this representation is to assist users in better understanding an object undergoing rotation over time, as well as improving the accuracy of the manipulation of 4D geometry. The representation, referred to in this project as the *4D to 3D abstraction*, takes a 3D object to complement the main 4D object. The 3D object will mimic a 4D planar rotation (xw, yw or zw) with the 3D axial rotation (x, y or z axes; equivalent to the yz, xz and xy planes of rotation respectively) utilising the 3D axis involved with the four dimensional rotation. For example: if the main 4D object is rotating about the yw plane, the 3D counterpart will rotate about the y axis (or the xz plane). This is a decomposition of a 4D double rotation into its 3D and 4D elements. In order to render a window with a separate 3D object on screen, a second plane was spawned in Unity, applying the appropriate ray marched shader. A second camera facing the plane allowed for a window displaying the 3D object to be rendered to a UI panel. The 3D object can be configured such that it is the 3D shape analogous to the main 4D shape on screen, or the 3D shape may be independent from the main 4D shape. During the experiment, the 3D shape is configured to be a 3D cube. The advantage of using a cube comes with its consistent right angles. The 3D cube conveys reasonably well how far away from a right angle the 4D object is, with respect to the global axes, as shown in fig. 5.5.

The primary advantage of this representation is also a risk. The *4D to 3D abstraction*, as mentioned earlier, abstracts the 4D rotation to a 3D rotation. As a result, it becomes very quick to identify what rotations are occurring with the main 4D object. This can be helpful for beginners, but unless considered with a critical eye, runs the risk of not teaching users how rotations work in $\mathbb{R}^4$, and instead provides an answer to a question without a user considering why the 3D cross section of an object behaves in the way that it does. Whilst the following may not invalidate the extension, it is worth noting. The *4D to 3D abstraction* representation is not extendable across n dimensions. Every other representation thus far could in theory be used in any dimension greater than two. This abstraction however relies on their being three more rotations in four dimensions, that can be directly mapped to the three planes of rotation in 3D. Consider rotations in $\mathbb{R}^5$: there are ten planes of rotation. If you tried to map 5D rotations to 4D; only three of the four new planes of rotation could be considered. This representation is expected to provide an increase of accuracy in the manipulation of 4D objects, as well as correctly interpreting any rotations it may undergo over time. This representation does not however provide any more information about the geometry of the object being handled, unlike the aforementioned representations.

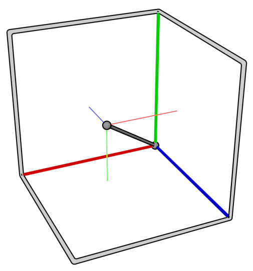
5.3 Rotation of a 4D Vector

As discussed in section 2.3.2, geometric algebra provides a platform for stable rotation of a vector in $\mathbb{R}^4$ without gimbal lock. The rotor is straightforward to work with when considering the abstract multivectors as single entities. Implementing a rotor in code, on the other hand, is a more complex task. A rotor is made up of three multivectors: A scalar, a bivector and a pseudo-scalar. To perform arithmetic operations on non-scalar multivectors, they must be broken down into projections onto their basis blades.

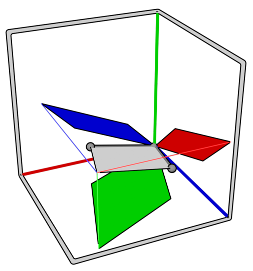
In $\mathbb{R}^4$ a grade 1 multivector (Vector) a can be broken down into four components. Each component is the vector projected onto one of the four axes (eq. (5.1)). One of the four projections is therefore a scalar value describing the vector's distance along an axis (fig. 5.6a).

$$a = a^x + a^y + a^z + a^w \tag{5.1}$$

In $\mathbb{R}^4$ a grade 2 multivector (Bivector) b is described by its area and orientation in space. A bivector can be broken down into six components, where each component is the bivectors



(a) The components of a vector



(b) The components of a bivector

Figure 5.6: *The projection of multivectors in $\mathbb{R}^3$ onto their basis blades. (a) shows the projections of a vector, parallel to the appropriate axes, stemming from the tip of the vector. (b) shows the projections of a bivector, constructed of two vectors, onto its basis planes.*

projection onto each plane that can be taken from the four axes (eq. (5.2)). A projection of the bivector onto a plane is a scalar value equivalent to the area of the projected bivector which will be less than or equal to the area of the bivector (fig. 5.6b).

$$b = b^{xy} + b^{xz} + b^{xw} + b^{yz} + b^{yw} + b^{zw} \tag{5.2}$$

As stated above, a rotor in $\mathbb{R}^4$ is comprised of a grade 0 multivector (Scalar) s, a bivector and a grade 4 multivector (pseudo-scalar) p (eq. (5.3)). The bivector defines the plane of rotation to rotate the vector about. The scalar describes how much to rotate the vector by. The pseudo-scalar is required for 4D rotation to normalise the rotor and ensure stable rotation; without it the vector may rotate unpredictably or increase/decrease in size.

$$R = s + b + p^{xyzw} = s + b^{xy} + b^{xz} + b^{xw} + b^{yz} + b^{yw} + b^{zw} + p^{xyzw} \tag{5.3}$$

$$R^{\dagger} = s - b - p^{xyzw} = s + b^{xy} - b^{xz} - b^{xw} - b^{yz} - b^{yw} - b^{zw} - p^{xyzw} \tag{5.4}$$

Bosch (2011) provides a code template written in C++ for a `Rotor4` in his initial blog post describing the rotor. Following his interactive article - an Introduction to Rotors - describing properties of geometric algebra in detail (Bosch b) Bosch provides a completed `Rotor3` class, compared against a Quaternion implementation in C++ (Bosch a). With this wealth of information, the foundations for the rotor could be implemented in Unity, C#. The 4D rotor however was incomplete. Whilst Bosch provided a completed `Rotor3` class, the $\mathbb{R}^4$ rotor-rotor product, and the $\mathbb{R}^4$ rotor-vector-sandwich product (the rotation of a vector) were not.

A multivector is considered as a sum of its projections, as shown above. To multiply two multivectors using the geometric product, the multivectors can be multiplied out given its distributive properties. As such, referencing a 4D geometric product multiplication table (Baker b), the subsequent multivector can be derived. eq. (5.5) shows how two multivectors can be multiplied using the geometric product with two vectors a and b in $\mathbb{R}^3$ using the appropriate geometric product multiplication table (Baker a).

$$\begin{aligned}
a &= a^x + a^y + a^z \\
b &= b^x + b^y + b^z \\
a * b &= +(a^x * b^x + a^x * b^y + a^x + b^z)e^x \\
&\quad + (a^y * b^x + a^y * b^y + a^y + b^z)e^y \\
&\quad + (a^z * b^x + a^z * b^y + a^z + b^z)e^z \\
a * b &= +(e + e^{xy} - e^{zx})e^x \\
&\quad + (-e^{xy} + e + e^{yz})e^y \\
&\quad + (e^{zx} - e^{yz} + e)e^z
\end{aligned} \tag{5.5}$$

Initially the equations for the rotor-rotor product and the rotation of a vector were derived by hand, using the same process as eq. (5.5). When testing the rotation of 4D objects however, there were consistent errors in their behaviour. Examples of these issues include: dramatic scaling, surface errors with ray marching, objects not rotating along the expected planes of rotation. These issues were focused around the vector produced from the rotor-vector sandwich product (eq. (2.2)).

Deriving the sandwich product by hand proved challenging, given that order matters as per the anti-commutative properties of multivectors. The process also produces grade 3 trivectors, which have influence over the rotated vector. `GAlgebra` is a package for symbolic geometric algebra calculus in python. The package was used to derive the appropriate equations for the rotor. This method of derivation proved quick and reliable, although translating the equations to C# proved very laborious. The code to produce the equations can be found in listing A.1. The final equations are grouped into their multivector components. The final equation to rotate a vector (eq. (5.1)) using a rotor (eq. (5.3)) and its reverse (eq. (5.4)) is grouped into the four components that make up the rotated vector. The equation to rotate a vector: eq. (A.1). The geometric product of two rotors a and b produces the equation (eq. (A.2)) and is grouped by a rotors eight components.

5.4 Methods of Rotation

The first challenge in rotating ray marched 4D shapes was passing the information to the shader. Unity's shader lab allows the programmer to implement and adjust properties of the shader. This is normally used for changing the colour or roughness. The shader properties used to manipulate ray marched objects included the (x, y, z, w) translation of the origin point, as well as the eight components of the Rotor used to rotate the object. eq. (A.1) was implemented in the shader to rotate any vector in the scene according to the eight rotor component shader properties.

Two methods of rotation were developed - only swipe based input was implemented. Swipe based input takes mouse delta values and keyboard inputs to rotate an object in the direction of mouse movement, allowing a user to restrict the plane of rotation through a keyboard input. Grab ball rotation was an attempt to expand upon the most commonly employed method of 3D rotation. Three arcs, or in the case of 4D: six hyper tori, each controlling a different plane of local rotation for the object.

Arcball or the virtual sphere can be used in 4D, as discussed in section 2.4. Swipe based rotation is simple to implement, and allows for restricting the planes of rotation. Therefore it was the minimal viable product in the development of an intuitive rotation mechanic. Whilst it would have been desirable to explore manipulation tools such as arcball or the virtual sphere, the allocated time versus the time of development did not allow for detailed exploration or development of such mechanics.

5.4.1 Swipe Based Input - Rotation About The Global Axes

Swipe based rotation mechanics are simple to implement and straightforward to understand. Restricting the plane of rotation allows users to be deliberate in the actions they take. A disadvantage of swipe based rotation in comparison with arcball or the virtual sphere, is requiring a secondary input to switch between dimensions.

Beginning with 3D swipe based rotation. Left clicking the mouse button and dragging it across the screen yields a delta value: a `Vector2` describing the difference in the x and y mouse positions between the current and previous frame. Using this information, the plane of rotation may be specified: an upward swipe (y) will rotate about the yz plane, and a sideways swipe (x) will rotate about the *xz* plane. Taking the mouse delta values and multiplying them by a `speed` constant allows them to be used as the input angle of rotation. The implementation allows a user to hold the `x` and `y` keys to restrict the plane of rotation about the *x* and *y* axes (*yz* and *xz* planes) respectively. To rotate about the *z* axis (*xy* plane), the user may hold the `z` key. To calculate the angle of rotation, the screen is divided into vertical and horizontal halves to produce a basic equation for translating circular motion to linear values. The user may then rotate the mouse in a circle about the center of the screen to produce the matching rotation.

The 3D rotation controls may be extended to four dimensions, keeping 3D rotation to the same as before; using the left mouse button. The right mouse button is then in control of 4D rotation. The mouse delta values are again used to control rotation in the *xw* and *yw* planes, respective to the *x* and *y* delta values. A user may restrict the planes of rotation using the same keys as before. Holding the `x` key will restrict the rotation to the *xw* plane, and holding the `y` will restrict the rotation to the *yw* plane. Holding the `z` key allows the user to rotate about the *zw* plane, where a user may move the cursor, again in a circular motion, to rotate the object.

Although the resulting rotation mechanic does not give the user control relative to the objects local axes, it received a warm reception during informal discussions with participants. It appears intuitive and easy to grasp. Separating the left and right click of the mouse to three and four dimensional rotations respectively, in combination with the ability to restrict the planes of rotation, allows users to be deliberate and purposeful with their actions.

5.4.2 4D Grab Ball – Rotation About The Local Axes

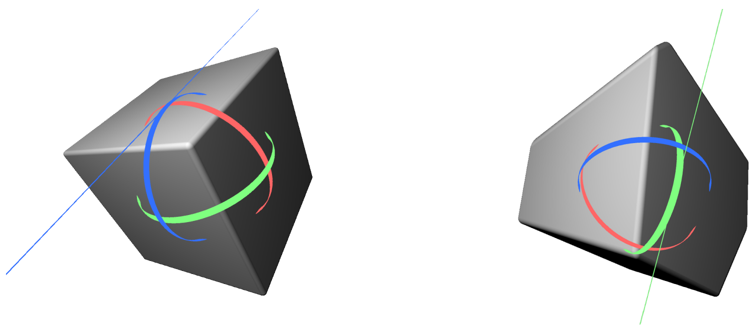
The most commonly employed method of controlling the rotation of an object, is with the use of a grab ball (as it is referred to in this paper). The grab ball provides a user with control over an object's rotation, relative to the object's local axis. The grab ball is well suited to a 2D interface, allowing a user to rotate an object without the need for a secondary input to switch dimensions.

In $\mathbb{R}^3$ dragging the mouse across each of the three arcs controls each of the three planes of rotation independently. Different software approaches the method of interaction in different ways. Unity's rotation tool appears to rotate geometry as the cursor follows the path tangent to the arc, from the point at which the user clicked on the arc. This tangent line is not indicated to a developer, and therefore rotation can sometimes feel rather clunky.

Initial exploration of the grab ball, began with an implementation in $\mathbb{R}^3$. As with Unity's implementation, the system was designed such that a rotation occurs as the cursor is dragged along the tangent relative to the point of contact with an arc. To improve upon Unity's implementation, a tangent line is rendered to the screen to guide the users actions, as shown in fig. 5.7a. To rotate the grab ball itself, Unity's `transform` was used,rotating the ball with the use of quaternions. After some improvements, the interaction in $\mathbb{R}^3$ seemed intuitive and promising.

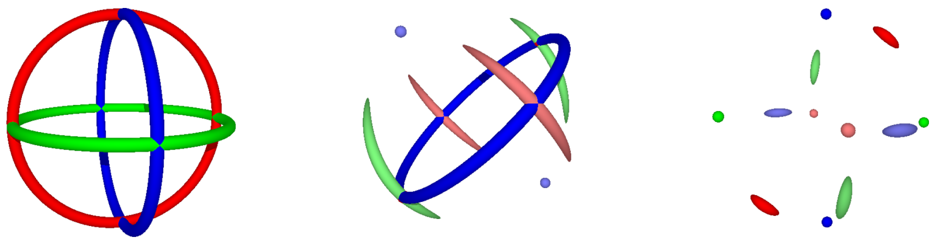
To rotate an object in four dimensions, a toggle was initially implemented, such that the three arcs would switch from the 3D planes of rotation, to the 4D planes of rotation (involving *w*). As the grab ball was rotated using a quaternion, it was held stationary when manipulating the geometry in 4D planes of rotation. This resulted in the first failure. Rotating in multiple planes of 4D rotation should effect the grab ball, and as a result the object and grab ball would no longer be in sync. This is shown in fig. 5.7b. Furthermore, a four dimensional grab ball would require six arcs, one for each plane of rotation.

Exploring the idea of the grab ball further, an SDF of a grab ball was developed: The union of six tori, each with their major radius spanning each of the six planes of rotation. To select an object, a ray cast is emitted from the cursor position on screen. A collider is required for the ray cast to detect the object it has interacted with. The ray marched shader has no collider and



(a) *Rotations in 3D* ***(b)*** *After rotations in 4D*

Figure 5.7: *Interactive mesh based arcs. When interacted with, an arc will overlay a tangent line which the path should follow. This works well for 3D rotation (a). When rotating in 4D, the grab ball remains stationary. As such, it becomes offset from the geometry (b).*

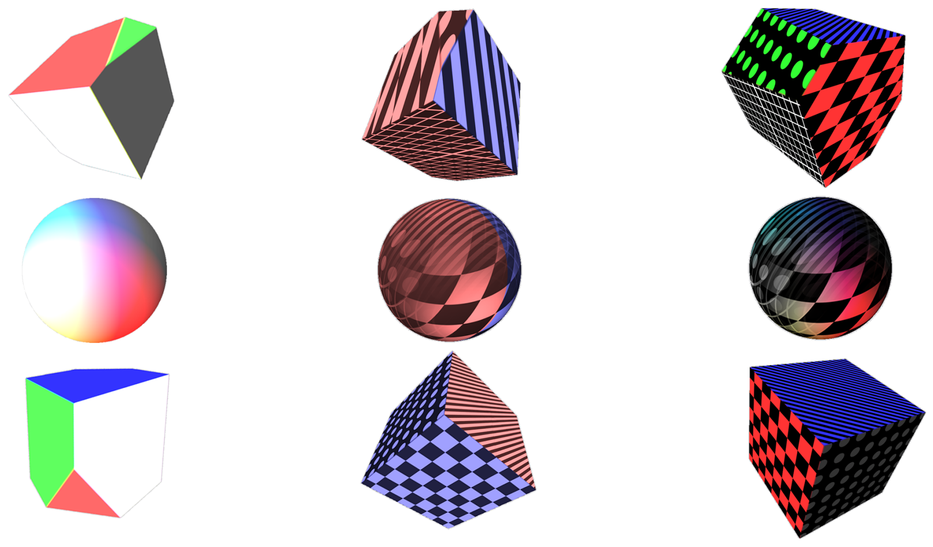


(a) *Not rotated in 4D* ***(b)*** *Small rotation in 4D* ***(c)*** *Large rotation in 4D*

Figure 5.8: *Ray marched grab ball concept using six tori. 4D arcs are lighter in colour to the 3D arcs. (a) shows the 4D arcs are hidden, and would be inaccessible. If rotated in 4D, the arcs begin to change, overtime indicating less and less what the result of interaction may be ((b), (c)).*

therefore cannot be interacted with, unlike the mesh based arcs. The ray marched grab ball however, does provide some insight. The 4D tori are still being viewed as a 3D cross section. As shown in section 5.4.2, rotating the grab ball in four dimensions morphs the arcs. As the arc may just appear as two spheres, it can become impossible to predict the plane of rotation the object may rotate in, when grabbing a particular arc. Furthermore, as shown in fig. 5.8a, as the arcs are the same size, and only a slice of them may be seen at a given time, often several of the six arcs are rendered invisible – obscured by other arcs.

Given more time for this investigation, research would be conducted into controlling the mesh of an arc through the use of a rotor, to rotate it in $\mathbb{R}^4$. It would be interesting to explore the shadow of a four dimensional grab ball, and how it may behave. Equally, it may be possible to define a collider synced to the ray marched grab ball. Explore how to control such an entity despite only a portion of it being visible at a given time, may prove more intuitive than one might expect.



(a) RGBW *(b) Directionally Coloured Pattern* *(c) RGBW Coloured Pattern*

Figure 5.9: *Quad-planar projection of colours and textures using the components of 4D normal vectors. (a), positive components are coloured x: Red, y: Green, z: Blue and w: white. Negative are dark and shaded with diffuse lighting. (b), patterns projected along each axis. negative components are coloured Blue, and positive are coloured Red. (c), patterns projected along each axis. Positive components coloured x: Red, y: Green, z: Blue and w: white. Negative are dark and shaded with diffuse lighting.*

5.5 Texturing Ray Marched Objects

Two cubes, offset by a multiple of 90° from each other will look identical. It is important to be able to distinguish the pose of two given symmetric objects of the same shape. Texturing the faces with colours and patterns allows a user to distinguish two otherwise identical shapes. Texturing the three dimensional cross section of four dimensional objects is a rather odd task given only a slice of the object can be seen at a given time. In ray marching, the common method of texturing objects is through normal based projection - also known as tri-planar projection (The_Art_of_Code 2020). Given an un-rotated object in $\mathbb{R}^3$, colours and textures can be projected onto the object according to the surface normal. The components of the normal vector can be mapped to the intensity of the projections. For example, multiplying the vector $(1, 0, 0)$ with a colour/texture would only project the texture along the x axis. Extending tri-planar projection to $\mathbb{R}^4$ utilises the 4D normal vector. Given the six planes of rotation, when a pattern is projected onto the cross section, it may morph under rotation. This makes some sense given a pattern is being projected from a direction we cannot see.

A common method of representing the surface normal in 3D is to map the three components of the normal vectors to the three primary colours: Red, Green and Blue. The origin point of the local axes exist in the center of the object. Therefore, components of the normal vector may be positive or negative. To distinguish all six sides of a shape in $\mathbb{R}^3$, if the vector component is positive, it may be coloured (RGB), otherwise it will be black. By adding the lighting to the surface of the object, the black sides of the object can be distinguished.

Three methods of distinguishing the shapes were explored in this project: *RGBW*, *Directionally coloured patterns* and *RGBW coloured patterns*; alongside the un-coloured diffuse lighting. *RGBW* (fig. 5.9a) builds upon the RGB colouration method of distinguishing faces. Given there are four components of the normal vector in $\mathbb{R}^4$, but only 3 primary colours, a surface facing the w axis will be coloured white. To more clearly emphasise the difference between the direction of a surface of an object, a pattern can be projected along each axis. Four different patterns were developed, and projected along each axes. To distinguish the positive and negative directions of the surface normal, the patterns must be multiplied by a colour. One colour scheme implemented was to colour the negative direction blue and the positive direction red (fig. 5.9b). The other was to mix the *RGBW* colour scheme with the patterns (fig. 5.9c).

5.6 Collecting Data

5.6.1 Data Format

Collected data was stored in a JSON format (listing A.2) for its ease of assembly and analysis. The Unity plugin: SimpleJSON (Bunny83) allows for easy reading and writing and of JSON data. The data was separated by tasks per representation. Each task was repeated several times per representations, with each task collecting data such as: time taken, participant answers, the pose of objects on screen, and participant survey responses held after each task. Data was not analysed prior to collection as analysis tools would be developed after the fact.

5.6.2 Remote Data Collection

Given the pandemic at the time of this project, it was necessary, for the purposes of collecting as much data as possible, that this experiment could function remotely. As stated, the project was built in Unity, allowing for easy building to WebGL, meaning the experiment could be hosted online. Unity's WebGL builds, and sites hosting WebGL builds, such as *itch.io* and *newgrounds.com*, implement security precautions making data collection a troublesome task. During runtime, the data is stored to a persistent data path. This worked well for local testing, however downloading the file from online hosting is not possible without setting up a database.

Collecting Data via Automatic Email

An attempt was made to send the data file through email upon the completion of the experiment. Unfortunately, some online hosting platforms, browsers and network configurations do not allow the sending of an anonymous email through a third party server for the sake of security.

Collecting Data via Copy & Paste

Unity enables a programmer to copy a string to the system clipboard. A button was implemented that would allow a user to copy the collected data to their clipboard. From there a user can manually email the data for collection. WebGL builds, however, do not allow the use of copying to a system clipboard. The Unity plugin: WebGLInput (kou yeung) allows for copying and pasting through the use of text boxes. The data was collected remotely from users via a copy/paste text box that would appear at the end of an experiment.

6 Evaluation

6.1 Aims

This investigation aims to evaluate the effectiveness of each of the three extensions to the 3D cross section of 4D geometry. Each extension has the potential to assist in a viewers understanding with respect to different aspects of geometry. To reiterate section 1.4: The evaluation focuses on which representation, if any, is most effective at conveying the surface geometry, the rotational pose, or assisting a users in manipulate geometry in $\mathbb{R}^4$. These will be measured through participants accuracy, confidence and time to answer, among other metrics.

6.2 Preliminary Research

6.2.1 How to Teach Others

To properly take advantage of, and evaluate each of the developed extension to the 3D cross section, a participant is required to understand the foundations of 4D space. As expected, many participants had little or no experience in 4D geometry, as per the responses to questionnaire (A.3). The tutorial video was structured with the goal of effectively teaching inexperienced participants prior to the experiment (section 4.1). The script for the video was drafted and tested several times with peers to tune the descriptions and explanations of complex topics to be clear and concise. The explanation for the rotation of geometry within and outside of the dimensions of the cross section is a notable example of a portion of the script that underwent several rewrites before viewers felt it was adequate.

During the development of each representation, preliminary evaluations with peers were conducted to see if each extension had potential in assisting users in their interpretation of geometry. As discussed, the *onion skin* representation proved unintuitive and too complex to digest for inexperienced users. Peers favoured the *w-axis timeline* as a less confusing alternative. As such, only three of the four developed representations were carried forward to experimentation.

6.2.2 Repeated Preliminary Experiments

The system was developed over the foundation that it was to be repeatedly tested and improved. In order to gather the best data, the experiment should not rush the user more than necessary. The experiment should also not last so long that it wears the participant down and affects their performance. To collect as much good data as possible, the timing of the experiment could not require an unreasonable commitment from potential participants. As such, the experiment was repeatedly tested and balanced to provide a user with an experience that allowed them to take their time, but did not last so long that it may wear them out, or deter people from participation.

For the experiment to be successful it must provide clear explanations of the experimental process and components participants will be interacting with, such as the extensions to the 3D cross section. Every piece of text was proof read by peers with little experience, to ensure the experiment was clear and concise throughout.

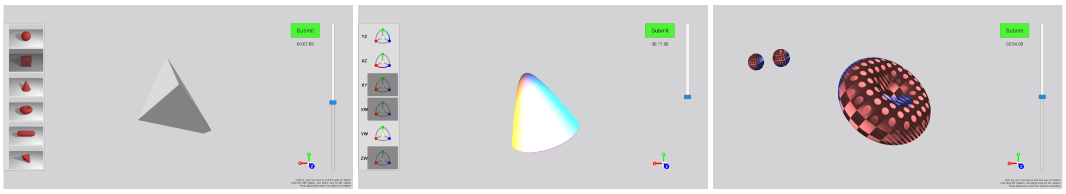

(a) Shape match *(b) Rotation match* *(c) Pose match*

Figure 6.1: User Interface layout for each task using the Control representation.

6.3 Tasks and Parameters

The experiment was separated into a series of repeated task. Each task would be repeated several times for each representation. Each task would try to capture the representations effectiveness at conveying a different aspect of geometry.

6.3.1 Shape Matching

The goal of shape matching is to evaluate a representations effectiveness at conveying the surface geometry of a 3D cross section of a 4D object. Given a random object, in a random pose, positioned randomly along the *w* axis, a user must identify the shapes 3D analogous counterpart. A user may select the shape they believe to be on screen using a menu on the left. (fig. 6.1a). To encourage users to interact with the object, and not make premature assumptions, the cone and capsule were biased such that their cross sections would appear as a sphere 35% of the time. The participant was allotted twenty seconds to play with and identify the shape on screen.

The cross section of different objects, such as the hyper-cube and pentachoron, may appear similar. When textured, however, have distinct colours or patterns, making them more easily identifiable. As such, the objects on screen were coloured exclusively white with diffuse lighting so as to avoid bias where a user may identify a relationship between the patterns and shapes. All of the developed shapes were included in this task, summing to six 3D objects, analogous to seven 4D objects.

After each attempt at the task, the participant was asked how confident they were in their answer. The participant was also asked if they felt they could relate what they saw in the 3D cross section of a 4D object to an analogous 3D to 2D example. These metrics would assist in differentiating blind guesses from an actual understanding.

6.3.2 Rotation Matching

The goal of rotation matching is to evaluate a representations effectiveness conveying to a user the rotations a 4D object may undergo; allowing a user to identify any active planes of continuous rotation. A given random object may rotate in up to three planes of rotation at once. There is a 100% chance of at least one 4D rotation (involving the *w* axis), approximately a 33% chance of a second 4D rotation, and a 30% chance of a 3D rotation. A user may select from a menu on the left hand side all of the planes of rotation they believe may be active. (fig. 6.1b). The participant was informed prior to this task that there may be between one and three planes of rotation, with a maximum of one 3D rotation at a given time. The participant was allotted sixty seconds to play with and identify the shape on screen. The sphere was not included for this experiment as its cross section does not change shape under rotation. Furthermore, all objects were textured given it may be impossible to identify a rotation if a symmetric objects surfaces are otherwise identical.

As with the shape matching task, after each attempt the participant was quizzed on their confidence and understanding.

6.3.3 Pose Matching

The goal of pose matching is to evaluate a representations effectiveness in assisting a user when manipulating geometry. Given a primary object in the same on-screen position as with the other tasks, and a secondary randomly oriented object at the top of the screen, a user must rotate the primary object to match the pose of the secondary object. (fig. 6.1c). The secondary object is randomly rotated exclusively along the 4D planes of rotation. Rotating the primary object using 3D rotation will simultaneously rotate the secondary object in 3D. The purpose of this is to allow a user to check all sides of the two shapes match, otherwise they may be restricted to looking at a single side of the secondary object, with no hint of similarity between the two objects. The participant was allotted three minutes to manipulate the shapes on screen to match their orientation as closely as possible. As with rotation match, the sphere was not included. All objects utilised a pattern texture to correctly identify the similarity between the two objects.

After the completion of each attempt the user was asked to rate the difficulty of the task. The difficulty may indicate if some shapes are easier to manipulate, and if a representation appears to assists users in the manipulation of geometry.

6.3.4 Task Format

It was expected that as users learned the behaviour of 4D shapes, they would improve. To avoid this bias leaking into the data, representations were presented in a random order. A control representation was included, without any extension to the 3D cross section being utilised.

Each representation would be subject to five iterations of shape matching, five iterations of rotation matching, and three iterations of pose matching, respectively. After each representation, the user would be shown graphs detailing their accuracy. The purpose of this was to keep participants engaged and attentive.

6.4 Limitations

The experimentation process for this investigation suffers from limitations that may limit the quality of the data collected. A notable drawback is the lack of time per task. Given the need to limit the time of a single experiment, time limits were placed on each task. The twenty seconds alloted to the shape match task proved too little, resulting in participants failing to answer in time (fig. 6.2). For users with limited experience, there was a moderately steep learning curve over the period of time an experiment was run for. Data will be varied based on different participants rate of learning, and as such, there is likely not enough evidence to draw any firm conclusions. Furthermore, the data may be biased against participants who participated remotely due to poor performance when the application is run in WebGL. The w-axis timeline, and multi-rotational view both have to render a fairly intensive scene, and as such tend to lag in browser.

6.5 Results

The following data is compiled from a total of 11 participants. All participants remained anonymous and provided explicit permission for their data to be analysed. Of the 11 participants, 5 attended in person. The remaining 6 participants attended the experiment remotely, and as such were subject to potentially limited browser performance. During the experiment participants were asked to answer questionnaire A.3. When asked how a participant learns best; 5 participants identified as a kinesthetic learner, 4 as a visual learner, and the remaining 2 learn best through reading and writing. Of the 11 participants, 3 had never heard of four dimensions before. 5 participants had briefly explored the concept through a personal interest. The remaining 3 were familiar with higher dimensionality in the form of data science and mathematics.

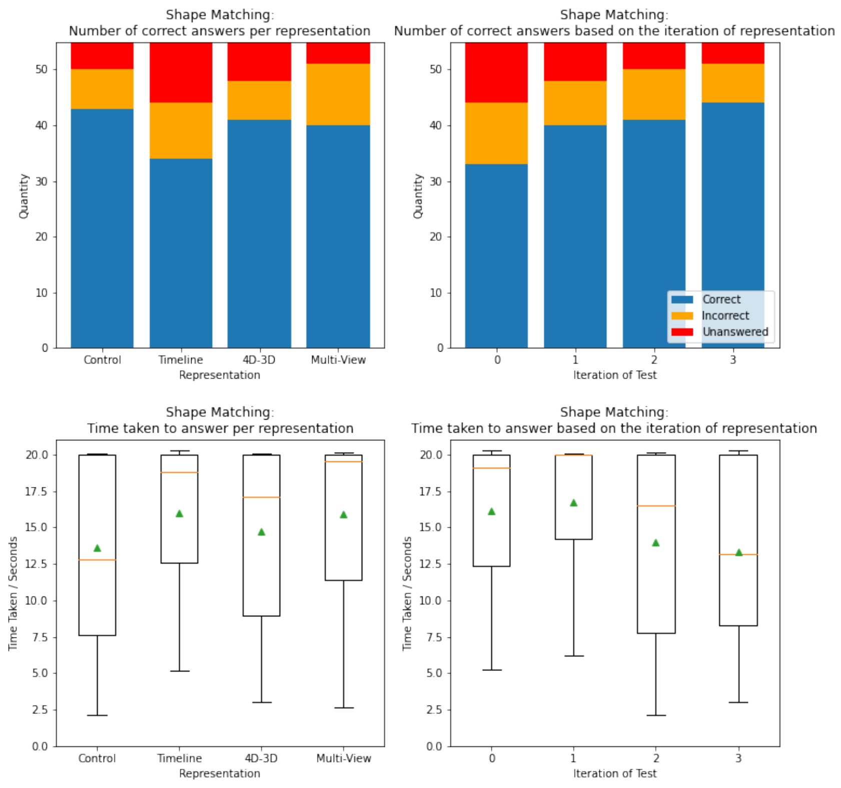


Figure 6.2: Shape matching correctness and timings

6.5.1 Quantitative Results

Shape Matching

Each participant become more familiar with the geometry of 4D shapes as they spent more time with them. As shown in fig. 6.2, participants answered more accurately over time; where each iteration follows the introduction of a new representation in random order. Following this, fig. 6.2 shows that participants were quicker to answer with each iteration of a new representation.

What is surprising; participants were most correct when using the control representation, and performed worst with the w-axis timeline. Whilst running the experiment in browser, the performance would decrease as the complexity of a ray marched scene increased. Furthermore, through discussion with participants, it appears a common theme that participants performed worse if there was more content to digest. These two elements are backed up by the time taken to answer per representation (fig. 6.2), where participants were much more likely to expire the twenty second time limit when interacting with representations rendering more content.

Encouragingly, a positive correlation can be seen between a participants correct answers and their confidence and understanding of their submission (fig. 6.3). Their confidence was rated on a scale of 0 to 10, and their understanding was a binary answer: "yes" or "no". This correlation suggests that participants were not making blind guesses. A strong correlation can also be seen between a participants confidence and how long they took to answer the question; suggesting a participant took longer to submit their answer if they were not confident in it. Following this, a decrease in correctness and understanding correlates with an increased time to answer (fig. 6.3).

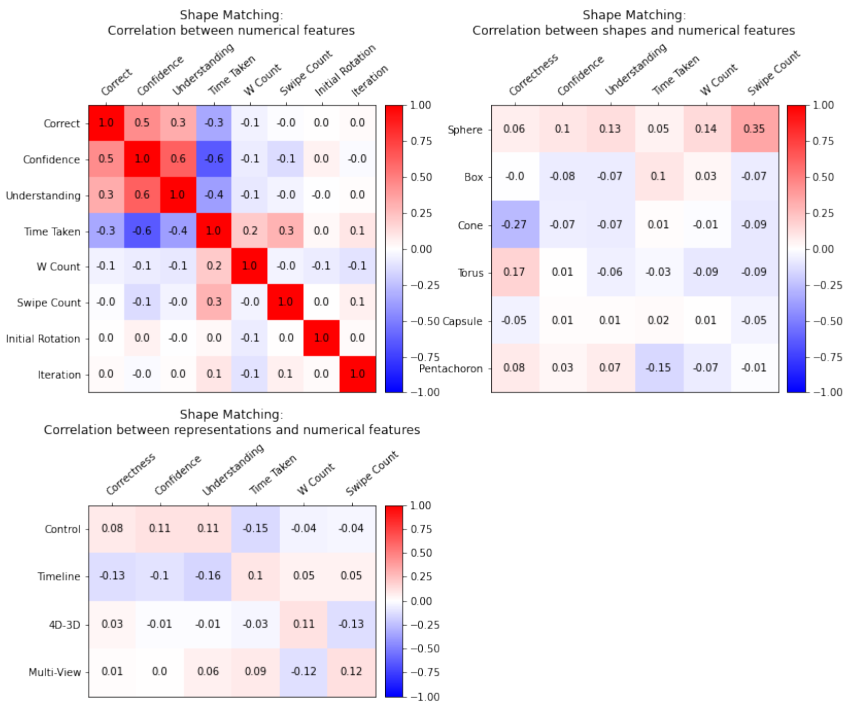


Figure 6.3: Shape matching correlations. "W Count": number of interactions with the w axis slider. "Swipe Count": number of rotations applied to the object. "Initial Rotation": The distance from 90° of the angle of the initial pose relative to no rotation

As stated in section 6.3, the cone and capsule had the potential to appear as a sphere. This may be a potential reason as to why participants were less likely to be correct when tasked with identifying the cone (fig. 6.3). On the other hand, users rarely incorrectly identified a torus. This is unsurprising as the torus has a very distinct shape. As expected, users would interact far more with the sphere, to indeed verify it was a sphere, given it will not change under rotation. This is indicated by the increased swipe count, used to rotate objects on screen (fig. 6.3).

Users tended to be less confident when using the w-axis timeline, and more confident whilst handling the control representation. This goes hand in hand with the correctness of a participants submission. Interestingly, participants tended to rotate objects more when using the multi-rotational view representation, despite already viewing objects from alternate perspectives. Participants also used the w-axis slider less for this representation. On the other hand, participants tended to use the w-axis slider more, and rotate the object less when using the 3D abstraction of 4D rotation. Given the 4D to 3D abstraction does not show any more information about the objects surface than the control representation, it is odd users handled it differently (fig. 6.3).

Rotation Matching

The active planes of continuous rotation were recorded using a 1×6 boolean array. Two boolean arrays were collected: The actual active planes of rotation, and the planes of rotation a participants suspected to be active. The similarity of two boolean arrays, A and B, can be calculated using the jaccard index eq. (6.1), (Glen 2022).

$$J(A, B) = \frac{|A \cap B|}{|A \cup B|} \tag{6.1}$$

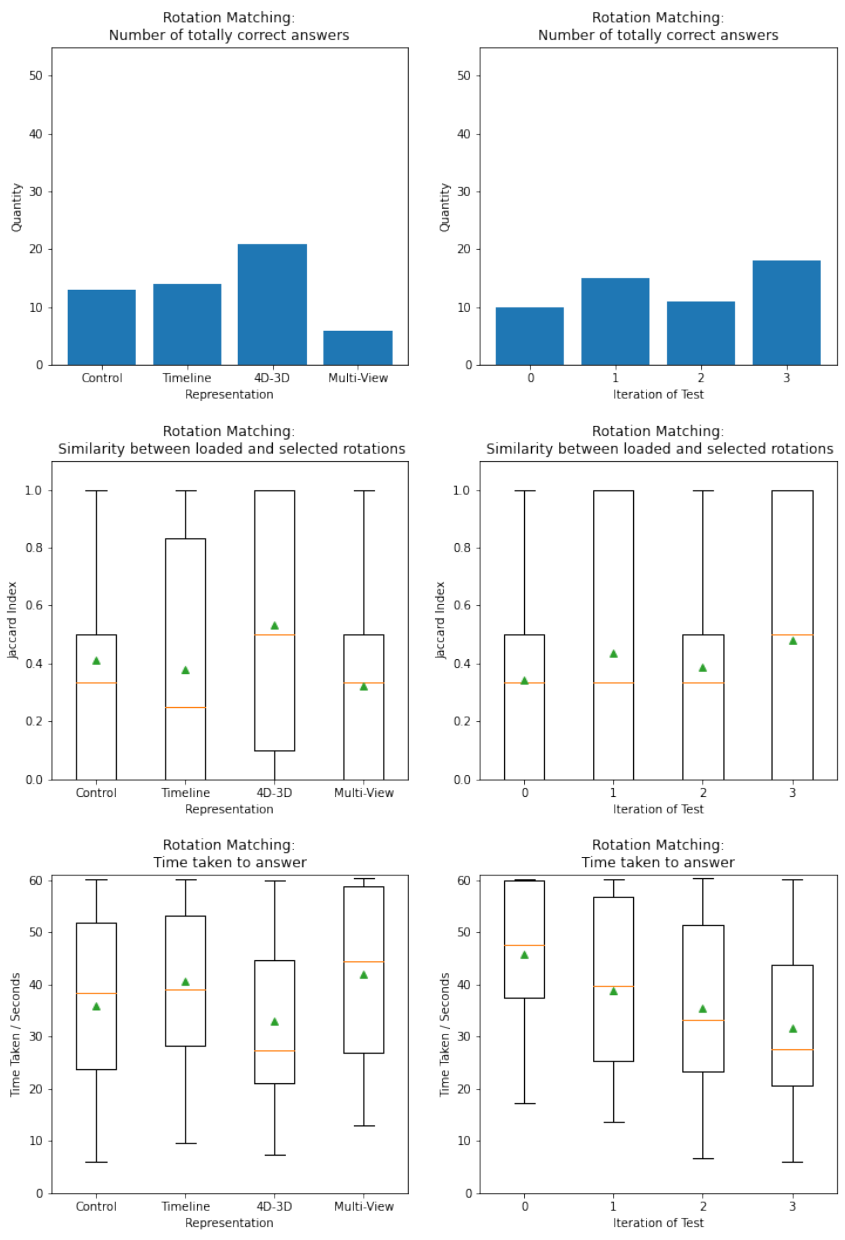


Figure 6.4: *Rotation matching correctness and timings*

With each iteration of a new representation, there appears to be a slight increase in accuracy, indicated by the quantity of perfect answers, and the similarity using the jaccard index (fig. 6.4). Furthermore, the time taken to answer clearly decreases as users spent more time with 4D shapes.

Participant performance when using the 4D to 3D rotational abstraction proved better than any other representation for all three metrics: totally correct answers, similarity of an answer, and the time taken to answer (fig. 6.4). Furthermore, participants were more confident in their answers whilst using this representation. Although this representation proved most effective in assisting users to identify the active planes of rotation, there is a debate as to whether this representation just gave the answer away, or if participants critically considered what they were seeing on screen.

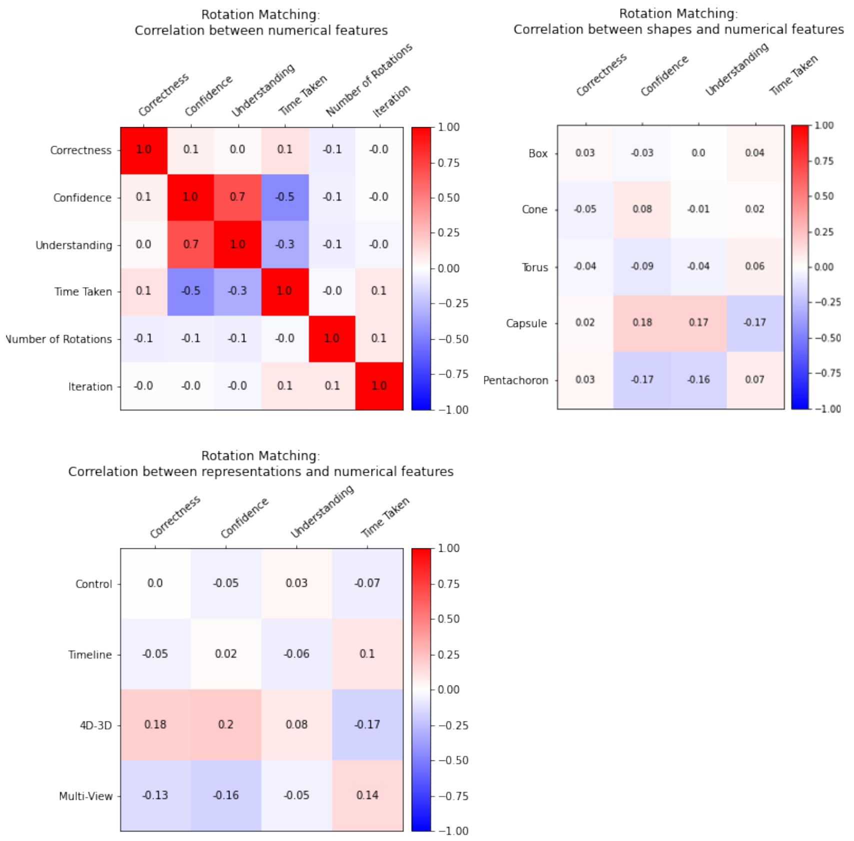


Figure 6.5: Rotation matching correlations. "Number of Rotations": The number of active planes of continuous rotation.

On the other hand, the multi-rotational view representation has a notably lower quantity of totally correct answers compared to the other representations. This trend can also be seen when analysing the similarity of users answers (fig. 6.4). Moreover, participant confidence is generally worse whilst using this representation (fig. 6.5). Discussion with participants yielded a common theme: Given four objects rotating on screen, where some rotate in 3D and some in 4D, participants were often mislead, seeing patterns in the rotation of different cross sections, but drawing incorrect conclusions. Participants also took longer to answer when using the multi-view representation. It seems that users require more time and experience with this representation to take advantage of the behaviour exhibited by each perspective.

The w-axis timeline did not seem notably better or worse with comparison to the control representation. The timeline representation does not provide any extra information on the rotations of a 4D object, and as such has not seemed to benefit users in interpreting the active planes of continuous rotation.

Unlike with the shape matching task, there is very little correlation between a participants level of confidence; measured on a scale of 0 to 10, and the correctness of their answer; measured using the jaccard index. This suggests that participants were likely, to some degree, guessing rather than being sure of their answers. Participants did show that they felt they had a level of understanding (rated on a trinary scale of "yes", "sort of" and "no") that correlated with their level of confidence in their answers (fig. 6.5).

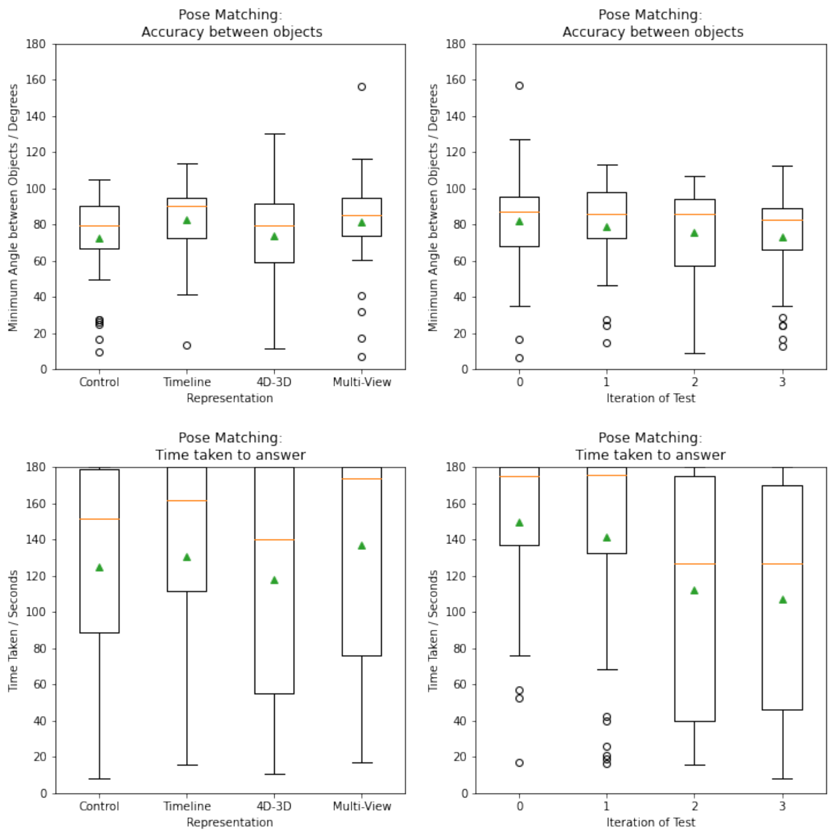


Figure 6.6: *Pose matching correctness and timings. Participants often expired the full 180 seconds, hence the box plots are capped at 180 seconds.*

As might be expected, participants confidence in their answers tended to decrease as they took longer to answer (fig. 6.5). It is likely that if a participant could not identify the active planes of rotation early on, they felt much less confident in their answer.

Generally a participants correctness was not affected by the shape on screen, although (fig. 6.5) suggests that users felt more confident whilst handling the capsule, and less so when dealing with the pentachoron. Interestingly however, participants took longer to answer when the object under rotation was the capsule.

Pose Matching

The accuracy of a participants submitted pose of a primary object, relative to the final orientation of a secondary object, is measured by the smallest angle of rotation between the two (algorithm 2).

Looking at fig. 6.6, the accuracy participants achieved over each representation followed, roughly, a normal distribution. There is some skewness in the data showing some of the most accurate submissions to be outliers. The average minimum angle, indicating accuracy, sits around 80° for each representation. No representation appears definitively better than another, although there is likely not enough data to draw any firm conclusions. From what can be seen in fig. 6.6, the 4D to 3D rotational abstraction has the highest variance and the control representation has the lowest variance. This may suggest that the representations do have some impact on a participants performance, but again, there is most likely not enough data to say for sure.

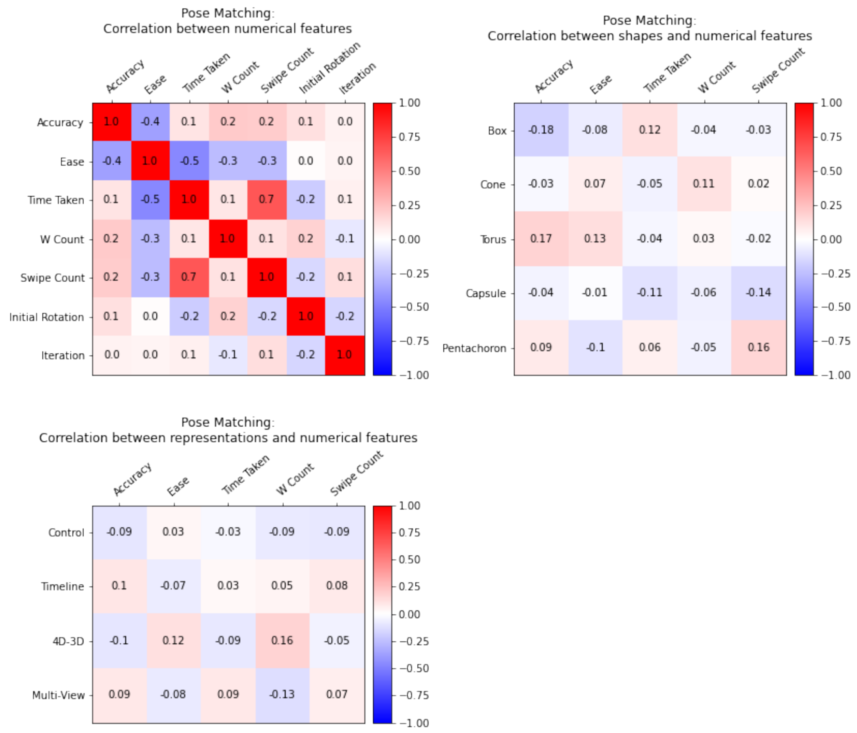


Figure 6.7: Pose matching correlations. "W Count": number of interactions with the w axis slider. "Swipe Count": number of rotations applied to the object. "Initial Rotation": The distance from 90° of the angle of the initial pose relative to no rotation.

```
Data:
    $v$ a random vector;
    $R_a$ the rotor defining the pose of one object;
    $R_b$ the rotor defining the pose of a second object;
Result:
    $\theta$ Minimum angle required to rotate one entity to another. $0 < \theta < \pi$
begin
    $v_a = R_a v R_a^\dagger$
    $v_b = R_b v R_b^\dagger$
    $\theta = \arccos\left(\frac{a \cdot b}{|a||b|}\right)$
end
```

Algorithm 2: Obtain the minimum angle required to rotate one entity to another

With each iteration of a new representation, it can be seen that users perform a little worse earlier on, and submit their answer faster after gaining practice over time (fig. 6.6). After submitting their final pose, each participant is asked to rate how easily they felt they were able to match the pose of the two objects (rated on a scale of 0 to 10). As might be expected, if a participant had to interact with the object less, and therefore spend less time manipulating the object, participants would rate the challenge to be easier (fig. 6.7). Counterintuitively, there is a fairly strong correlation suggesting that a users accuracy increased if they found a challenge to be more difficult. This could be the case if a participant believed a shape to be in the correct pose, but was in fact far from it. Another suggestion could be that a participant spent more time on a shape in a difficult pose, which subsequently lead to an increase in accuracy following an increase in time spent manipulating the objects. This, however, is speculation.

As shown in fig. 6.7, the more time a user spent manipulating the object the more accurate they were. If the initial pose of the randomly oriented secondary object was closer to 90°, the participants tended to spend more time interacting with the object, and generally rated it a harder challenge to match the orientation of the two objects.

The 4D hyper-cube proved the hardest shape to manipulate, with participants scoring lower accuracies when dealing with it. Participants also took more time to submit their final pose compared to the other shapes. On the other hand, participants were more accurate, and found it easier to manipulate the torus. The capsule had faster submission times and required less interaction for participants to be satisfied with the pose of the shape. The pentachoron was found to be the most difficult object to manipulate, and required more interaction that any other.

6.5.2 Qualitative Results

At the end of the experiment, participants were asked to complete the final section of the questionnaire (section A.3) which they had started at the beginning of the experiment. This stage of the questionnaire probed users on their opinions of each extension to the 3D cross section that they had interacted with during the course of the experiment. The summary of their opinions are listed below.

W-axis Timeline

The w-axis timeline is a "simple presentation of geometry" that helps "gain a visual understanding of what a 4D object is". Participants found that the representation saved time in identifying shapes, and informed users about what will happen before interacting with the w-axis scroll bar.

Some participants noted that this representation does not provide any assistance in conveying the "condition of the shape during rotation". Others noted that the w-axis timeline overloads users with "too much information". As stated, the complexity of the scene also results in a performance drop whilst running the experiment in a browser.

Multi-rotational View

Participants noted that this representation explained the concept of 4D "in an interesting way". Some felt this representation helped them distinguish different planes of rotation.

The general consensus, on the other hand, follows that this representation appeared cluttered, with too much information that participants felt was often overwhelming. Given each cross section would exhibit different behaviour under rotation, participants often incorrectly identified patterns, leading to misinterpretation of the rotations a shape would undergo. With four shapes on the screen, the amount of information was somewhat overwhelming. Like the w-axis timeline, this representation would occasionally lag when loaded in browser.

4D to 3D Abstraction

This representation was considered "the easiest to work with" and more simple to understand in comparison to the other representations. Participants also found it assisted them to distinguish several planes of rotation.

Outside of the rotation match challenge, many participants did not make much use out of the extension. Many appreciated the simplicity with their being less information on the screen, but did not find the extension helpful in a setting other than identifying active planes of rotation. During the pose matching task, participants felt that the rotations the cube would undergo were confusing or seemingly unrelated to the rotations they would execute on the main object.

6.5.3 Analysis and Discussion

From the collected quantitative data from the experiments, combined with the qualitative data collected via the experimental survey, it can be concluded that: without an adequate foundation in the understanding of higher dimensional geometry, the benefits of some extensions to the 3D cross section cannot be fully realised. As proposed in the van-Hiele model of geometric thinking (Šafránková 2012), a student cannot progress to level N without first having completed level $N - 1$. It is likely that extending a 3D cross section of a 4D object, before users have mastered the behaviour of the standard 3D cross section is what resulted in such feelings of being overwhelmed with information.

As participants interacted with objects more, their accuracy increased. Removing the boundary of a time limit would very likely yield better data. Such an experiment would require volunteers to commit a relatively large portion of time, in order for them to build an adequate foundation, before evaluating these extensions further. Running such an experiment, however, would likely provide better data, allowing for deeper analysis into the advantages and disadvantages of each proposed representation.

We can at the very least, draw the conclusion that the data has some validity given the correlation between a participants confidence and understanding, with their accuracy. This suggests that participants were not just guessing blindly, and, as such, the data is indicative of what may occur in a larger scale experiment.

Extensions to the 3D Cross Section

The duration of a single experiment was relatively short. Even with a tutorial video, participants generally had very limited experience with 4D space. As such, complex extensions, such as multi-view, caused participants to feel overwhelmed with information. With no experience with such representations prior to the experiment, participants were, for the most part, unable to take advantage of the benefits they offer. Confidence was higher when utilising a representation that displayed less information at a given time, and generally participants were more accurate, and answered in less time. The experiment would likely have benefited if participants were provided with time to explore each representation prior to evaluating it through testing.

The 4D to 3D abstraction successfully allowed users to differentiate 3D and 4D rotations. Given its low overhead, it was not overwhelming for novice users. However, this representation is limited by its lack of versatility. Participants commented that they very rarely used the extension outside of the rotation matching task. The 4D to 3D abstraction could be used in tandem with another extension, but this approaches the problem of overloading users with too much information.

Participants seemed to prefer the w-axis timeline representation, despite it lagging in browsers. The quantitative data however, does not indicate any benefit to users in identifying the surface geometry of 4D shapes with comparison to the control representation. The shape matching task, where this representation was expected to show a benefit, had a grossly underestimated time limit. Perhaps with more time, different results may have been uncovered.

Of the three representations, none seemed to benefit users in the manipulation of 4D geometry in a notable way. From the questionnaire and through discussion with participants, it would seem that as they were aiming to match the main object to a secondary object, they payed little attention to any other objects in the scene.

Performance Over Time

In each challenge there are clear improvements over time. As participants spent more time with the 4D geometry, their accuracy often increased, and their time to answer decreased. These trends are less notable for the shape matching task.

7 Conclusion

7.1 Summary

Over the course of this investigation, a system capable of rendering the cross sections of several 4D shapes, in real time, was developed with the use of ray marching. Manipulation techniques were experimented with, and a system to rotate geometry without instability or gimbal lock was employed. Three extensions to the 3D cross section were developed and experimentally validated. Experiments placed a core focus on teaching participants about the fourth dimension in order to equip them with the necessary knowledge to reliably evaluate these representations of geometry.

The concept of 4D geometry can be unintuitive to users with limited experience. Despite this, over a two hour period where the experiment was conducted, participants showed very clear signs of learning and improvement. Of the three extensions to the 3D cross section that were developed, it was found that: typically a representation of a shape is generally easier to interpret if there is less information for a user to digest. Participants were more confident and often more accurate when dealing with a representation that showed less information. It was also found that abstracting complex phenomena such as rotations in $\mathbb{R}^4$ assisted users when identifying active planes of rotation; although the proposed abstraction is not generalisable to higher dimensions.

Through the collected qualitative data (contrary to fig. 6.2) participants felt the w-axis timeline representation assisted in a users understanding of the geometry. During experimentation however, the multi-rotational view proved very limited to novice users. Participants felt overwhelmed by the information presented, which often lead to misinterpretation of the behaviour exhibited whilst an object was under continuous rotation. The 4D to 3D rotational abstraction appears promising, although is limited in use. It is likely better suited for teaching, than for practical use. All in all, the investigation was limited by time, and by the quantity of data that could be collected. As it stands, however, there appears no single best extension to the 3D cross section when interpreting the surface of, or manipulating 4D geometry.

7.2 Future Work

In order to collect as much data as possible in a limited time, experiments could not require an unreasonable commitment from participants. With more time to conduct an experiment, the investigation may have been able to properly take advantage of the van-Hiele model to produce more conclusive results. Future works might include composing an experiment that considers a progressive teaching experience, allowing participants to build a satisfactory foundation of four dimensions, prior to evaluating each extension.

Other areas of interest include the development of a system to render mesh based geometry, allowing for the creation of more complex shapes. Alongside this, further exploration into intuitive manipulation mechanics would be desirable. This may include smarter gesture recognition to assist users, or implementing a double arcball (Shoemake 1994), to more precisely manipulate 4D geometry. Finally, it could be of interest to conduct research into more complex visualisation methods such as 4D stereographic projection, and quantifiably compare it to the 3D cross section.

A Appendices

A.1 Rotor Equation Derivation

```python
from sympy import symbols
from galgebra.ga import Ga
from galgebra.printer import Format

Format(Fmode = False, Dmode = True)

s4coords = (x,y,z,w) = symbols('x y z w', real=True)
s4 = Ga('e', g=[1,1,1,1],
coords=s4coords)

def rotate_vector():
  # Vector
  a = s4.mv('a','vector')

  # Rotor
  s = s4.mv('s', 'scalar')
  b = s4.mv('b','bivector')
  p = s4.mv('p', 'pseudo')
  rotor = s + b + p

  (rotor * a * rotor.rev()).Fmt(3)

def rotor_rotor_product
  # Rotor A
  a_s = s4.mv('a_s', 'scalar')
  a_b = s4.mv('a_b','bivector')
  a_p = s4.mv('a_p', 'pseudo')
  rotor_a = a_s + a_b + a_p

  # Rotor B
  b_s = s4.mv('b_s', 'scalar')
  b_b = s4.mv('b_b','bivector')
  b_p = s4.mv('b_p', 'pseudo')
  rotor_b = b_s + b_b + b_p

  (rotor_a * rotor_b).Fmt(3)
```

Listing A.1: *Rotor4 product derivation using Geometric Algebra in python:*
Rotor4-Vector rotation using double reflection; producing eq. (A.1).
Rotor4-Rotor4 product to apply a 4D Rotor to another 4D Rotor; producing eq. (A.2).

$$
\begin{aligned}
&+ \Big(2a^{w}b^{xw}s + 2a^{w}b^{xy}b^{yw} + 2a^{w}b^{xz}b^{zw} + 2a^{w}b^{yz}p^{xyzw} \\
&\quad - a^{x}(b^{xw})^2 - a^{x}(b^{xy})^2 - a^{x}(b^{xz})^2 + a^{x}(b^{yw})^2 + a^{x}(b^{yz})^2 + a^{x}(b^{zw})^2 - a^{x}(p^{xyzw})^2 + a^{x}s^2 \\
&\quad - 2a^{y}b^{xw}b^{yw} + 2a^{y}b^{xy}s - 2a^{y}b^{xz}b^{yz} + 2a^{y}b^{zw}p^{xyzw} \\
&\quad - 2a^{z}b^{xw}b^{zw} + 2a^{z}b^{xy}b^{yz} + 2a^{z}b^{xz}s - 2a^{z}b^{yw}p^{xyzw}\Big)\boldsymbol{e}_x \\
&+ \Big(- 2a^{w}b^{xw}b^{xy} - 2a^{w}b^{xz}p^{xyzw} + 2a^{w}b^{yw}s + 2a^{w}b^{yz}b^{zw} \\
&\quad - 2a^{x}b^{xw}b^{yw} - 2a^{x}b^{xy}s - 2a^{x}b^{xz}b^{yz} - 2a^{x}b^{zw}p^{xyzw} \\
&\quad + a^{y}(b^{xw})^2 - a^{y}(b^{xy})^2 + a^{y}(b^{xz})^2 - a^{y}(b^{yw})^2 - a^{y}(b^{yz})^2 + a^{y}(b^{zw})^2 - a^{y}(p^{xyzw})^2 + a^{y}s^2 \\
&\quad + 2a^{z}b^{xw}p^{xyzw} - 2a^{z}b^{xy}b^{xz} - 2a^{z}b^{yw}b^{zw} + 2a^{z}b^{yz}s\Big)\boldsymbol{e}_y \\
&+ \Big(- 2a^{w}b^{xw}b^{xz} + 2a^{w}b^{xy}p^{xyzw} - 2a^{w}b^{yw}b^{yz} + 2a^{w}b^{zw}s \\
&\quad - 2a^{x}b^{xw}b^{zw} + 2a^{x}b^{xy}b^{yz} - 2a^{x}b^{xz}s + 2a^{x}b^{yw}p^{xyzw} \\
&\quad - 2a^{y}b^{xw}p^{xyzw} - 2a^{y}b^{xy}b^{xz} - 2a^{y}b^{yw}b^{zw} - 2a^{y}b^{yz}s \\
&\quad + a^{z}(b^{xw})^2 + a^{z}(b^{xy})^2 - a^{z}(b^{xz})^2 + a^{z}(b^{yw})^2 - a^{z}(b^{yz})^2 - a^{z}(b^{zw})^2 - a^{z}(p^{xyzw})^2 + a^{z}s^2\Big)\boldsymbol{e}_z \\
&+ \Big(- a^{w}(b^{xw})^2 + a^{w}(b^{xy})^2 + a^{w}(b^{xz})^2 - a^{w}(b^{yw})^2 + a^{w}(b^{yz})^2 - a^{w}(b^{zw})^2 - a^{w}(p^{xyzw})^2 + a^{w}s^2 \\
&\quad - 2a^{x}b^{xw}s + 2a^{x}b^{xy}b^{yw} + 2a^{x}b^{xz}b^{zw} - 2a^{x}b^{yz}p^{xyzw} \\
&\quad - 2a^{y}b^{xw}b^{xy} + 2a^{y}b^{xz}p^{xyzw} - 2a^{y}b^{yw}s + 2a^{y}b^{yz}b^{zw} \\
&\quad - 2a^{z}b^{xw}b^{xz} - 2a^{z}b^{xy}p^{xyzw} - 2a^{z}b^{yw}b^{yz} - 2a^{z}b^{zw}s\Big)\boldsymbol{e}_w
\end{aligned}
\tag{A.1}
$$

4D rotor-vector sandwich geometric product closed form expression, grouped by vector components

$$
\begin{aligned}
&+ \left(-a_b^{xw}b_b^{xw} - a_b^{xy}b_b^{xy} - a_b^{xz}b_b^{xz} - a_b^{yw}b_b^{yw} - a_b^{yz}b_b^{yz} - a_b^{zw}b_b^{zw} + a_p^{xyzw}b_p^{xyzw} + a_s b_s\right) \\
&+ \left(-a_b^{xw}b_b^{yw} + a_b^{xy}b_s - a_b^{xz}b_b^{yz} + a_b^{yw}b_b^{xw} + a_b^{yz}b_b^{xz} - a_b^{zw}b_p^{xyzw} - a_p^{xyzw}b_b^{zw} + a_s b_b^{xy}\right) \boldsymbol{e}_x \wedge \boldsymbol{e}_y \\
&+ \left(-a_b^{xw}b_b^{zw} + a_b^{xy}b_b^{yz} + a_b^{xz}b_s + a_b^{yw}b_p^{xyzw} - a_b^{yz}b_b^{xy} + a_b^{zw}b_b^{xw} + a_p^{xyzw}b_b^{yw} + a_s b_b^{xz}\right) \boldsymbol{e}_x \wedge \boldsymbol{e}_z \\
&+ \left(a_b^{xw}b_s + a_b^{xy}b_b^{yw} + a_b^{xz}b_b^{zw} - a_b^{yw}b_b^{xy} - a_b^{yz}b_p^{xyzw} - a_b^{zw}b_b^{xz} - a_p^{xyzw}b_b^{yz} + a_s b_b^{xw}\right) \boldsymbol{e}_x \wedge \boldsymbol{e}_w \\
&+ \left(-a_b^{xw}b_p^{xyzw} - a_b^{xy}b_b^{xz} + a_b^{xz}b_b^{xy} - a_b^{yw}b_b^{zw} + a_b^{yz}b_s + a_b^{zw}b_b^{yw} - a_p^{xyzw}b_b^{xw} + a_s b_b^{yz}\right) \boldsymbol{e}_y \wedge \boldsymbol{e}_z \\
&+ \left(a_b^{xw}b_b^{xy} - a_b^{xy}b_b^{xw} + a_b^{xz}b_p^{xyzw} + a_b^{yw}b_s + a_b^{yz}b_b^{zw} - a_b^{zw}b_b^{yz} + a_p^{xyzw}b_b^{xz} + a_s b_b^{yw}\right) \boldsymbol{e}_y \wedge \boldsymbol{e}_w \\
&+ \left(a_b^{xw}b_b^{xz} - a_b^{xy}b_p^{xyzw} - a_b^{xz}b_b^{xw} + a_b^{yw}b_b^{yz} - a_b^{yz}b_b^{yw} + a_b^{zw}b_s - a_p^{xyzw}b_b^{xy} + a_s b_b^{zw}\right) \boldsymbol{e}_z \wedge \boldsymbol{e}_w \\
&+ \left(a_b^{xw}b_b^{yz} + a_b^{xy}b_b^{zw} - a_b^{xz}b_b^{yw} - a_b^{yw}b_b^{xz} + a_b^{yz}b_b^{xw} + a_b^{zw}b_b^{xy} + a_p^{xyzw}b_s + a_s b_p^{xyzw}\right) \boldsymbol{e}_x \wedge \boldsymbol{e}_y \wedge \boldsymbol{e}_z \wedge \boldsymbol{e}_w
\end{aligned}
\tag{A.2}
$$

4D rotor-rotor geometric product, grouped by rotor components

A.2 Data Collection: JSON Schema

```
{
  representation_name:
  {
    // repeated i times for each representation
    shape_match_i:
    {
      loaded_shape,
      selected_shape, // answer of suspected object
      time, // time taken
      initial_rotor, // object rotation: rotor components
      final_rotor,
      w_count, // interactions with the hyperplane slider
      swipe_count, // number of applied rotations
    },
    shape_match_i_survey
    {
      confidence, // users confidence
      understanding, // users understanding
    },
    rotation_match_i:
    {
      loaded_shape,
      loaded_rotation, // boolean array of active planes of continuous rotation
      selected_rotation, // answer of suspected planes of rotation
      time, // time taken
    },
    rotation_match_i_survey:
    {
      confidence, // users confidence
      understanding, // users understanding
    },
    pose_match_i:
    {
      loaded_shape,
      time, // time taken
      main_object_rotor, // final rotor components of main object
      match_object_rotor, // final rotor components of posed object to be matched
      initial_match_object_rotor, // initial pose of the object to be matched
      w_count, // interactions with the hyperplane slider
      swipe_count // number of applied rotations
    },
    pose_match_i_survey:
    {
      ease, // user rated ease
    },
  },
}
```

Listing A.2: JSON schema for data collection

A.3 Questionnaire

4D Shapes Questionnaire

An Investigation into the best way to represent a cross section of a 4D objects

* Required

Experiment Information

Ask joe for this information

1. User ID *

2. Date of Experiment *

   Example: January 7, 2019

3. Time of Experiment *

   Example: 8:30 AM

Before Experiment

4. Do you have any disability or visual impairment that may affect your comprehension of shapes or colours? (e.g colour blindness) *

5. How do you learn best? *

Mark only one oval.

- ⬭ Visual Learning (graphs or diagrams)
- ⬭ Auditory Learning (listening to an explanation)
- ⬭ Kinesthetic Learning (hands on tasks)
- ⬭ Reading/Writing

6. Have you heard of / researched 4D shapes in the past? *

Mark only one oval.

- ⬭ Yes
- ⬭ No

7. Briefly describe your experience with 4D shapes:

After Tutorial Video

8. Is there anything that came up in the tutorial video that you feel may be a challenge for you?

After Experiment

W-Axis “Timeline” Representation

9. What about this representation did you like? *

10. What about this representation did you dislike? *

Multi-Rotational View Representation

11. What about this representation did you like? *

12. What about this representation did you dislike? *

3D Axial Rotation to mimic the 3D axis component of 4D Rotation Representation (4D to 3D Rotation)

13. What about this representation did you like? *

14. What about this representation did you dislike? *

Permission

15. Do you give permission for the data collected in this experiment to be analysed? The data will be analysed and presented anonymously. You can change your mind at any time after the experiment. *

Mark only one oval.

- ⬭ Yes
- ⬭ No

A.4 Ethics Checklist

School of Computing Science
University of Glasgow

Ethics checklist form for 3rd/4th/5th year, and taught MSc projects

This form is only applicable for projects that use other people ('participants') for the collection of information, typically in getting comments about a system or a system design, getting information about how a system could be used, or evaluating a working system.

If no other people have been involved in the collection of information, then you do not need to complete this form.

If your evaluation does not comply with any one or more of the points below, please contact the Chair of the School of Computing Science Ethics Committee (matthew.chalmers@glasgow.ac.uk) for advice.

If your evaluation does comply with all the points below, please sign this form and submit it with your project.

---

1. Participants were not exposed to any risks greater than those encountered in their normal working life.

   Investigators have a responsibility to protect participants from physical and mental harm during the investigation. The risk of harm must be no greater than in ordinary life. Areas of potential risk that require ethical approval include, but are not limited to, investigations that occur outside usual laboratory areas, or that require participant mobility (e.g. walking, running, use of public transport), unusual or repetitive activity or movement, that use sensory deprivation (e.g. ear plugs or blindfolds), bright or flashing lights, loud or disorienting noises, smell, taste, vibration, or force feedback

2. The experimental materials were paper-based, or comprised software running on standard hardware.

   Participants should not be exposed to any risks associated with the use of non-standard equipment: anything other than pen-and-paper, standard PCs, laptops, iPads, mobile phones and common hand-held devices is considered non-standard.

3. All participants explicitly stated that they agreed to take part, and that their data could be used in the project.

   If the results of the evaluation are likely to be used beyond the term of the project (for example, the software is to be deployed, or the data is to be published), then signed consent is necessary. A separate consent form should be signed by each participant.

   Otherwise, verbal consent is sufficient, and should be explicitly requested in the introductory script.

4. No incentives were offered to the participants.

   The payment of participants must not be used to induce them to risk harm beyond that which they risk without payment in their normal lifestyle.

5. No information about the evaluation or materials was intentionally withheld from the participants.
   Withholding information or misleading participants is unacceptable if participants are likely to object or show unease when debriefed.

6. No participant was under the age of 16.
   Parental consent is required for participants under the age of 16.

7. No participant has an impairment that may limit their understanding or communication.
   Additional consent is required for participants with impairments.

8. Neither I nor my supervisor is in a position of authority or influence over any of the participants.
   A position of authority or influence over any participant must not be allowed to pressurise participants to take part in, or remain in, any experiment.

9. All participants were informed that they could withdraw at any time.
   All participants have the right to withdraw at any time during the investigation. They should be told this in the introductory script.

10. All participants have been informed of my contact details.
    All participants must be able to contact the investigator after the investigation. They should be given the details of both student and module co-ordinator or supervisor as part of the debriefing.

11. The evaluation was discussed with all the participants at the end of the session, and all participants had the opportunity to ask questions.
    The student must provide the participants with sufficient information in the debriefing to enable them to understand the nature of the investigation. In cases where remote participants may withdraw from the experiment early and it is not possible to debrief them, the fact that doing so will result in their not being debriefed should be mentioned in the introductory text.

12. All the data collected from the participants is stored in an anonymous form.
    All participant data (hard-copy and soft-copy) should be stored securely, and in anonymous form.

Project title Developing and Evaluating Extensions to 3D Cross Sections of 4D Geometry

Student's Name Joe Subbiani

Student Number 2377990s

Student's Signature [illegible]

Supervisor's Signature [illegible]

Date 29/03/2022

7 Bibliography

M. J. Baker. Maths - Clifford Algebra - Arithmetic, a. URL `https://www.euclideanspace.com/maths/algebra/clifford/algebra/index.htm`.

M. J. Baker. Maths - 4D Clifford Algebra - Arithmetic, b. URL `https://www.euclideanspace.com/maths/algebra/clifford/d4/arithmetic/index.htm`.

R. Balakrishnan, T. Baudel, G. Kurtenbach, and G. Fitzmaurice. The Rockin'Mouse: integral 3D manipulation on a plane. In *Proceedings of the ACM SIGCHI Conference on Human factors in computing systems*, pages 311–318, Atlanta Georgia USA, Mar. 1997. ACM. ISBN 978-0-89791-802-2. doi: 10.1145/258549.258778. URL `https://dl.acm.org/doi/10.1145/258549.258778`.

M. t. Bosch. Code Template for Rotor3 and Quaternion in C++, a. URL `https://marctenbosch.com/quaternions/code.htm`.

M. t. Bosch. Let's remove Quaternions from every 3D Engine, b. URL `https://marctenbosch.com/quaternions/`.

M. t. Bosch. 4D Rotations and the 4D equivalent of Quaternions, May 2011. URL `https://marctenbosch.com/news/2011/05/4d-rotations-and-the-4d-equivalent-of-quaternions/`.

M. t. Bosch. N-dimensional rigid body dynamics. *ACM Transactions on Graphics*, 39(4), July 2020. ISSN 0730-0301, 1557-7368. doi: 10.1145/3386569.3392483. URL `https://dl.acm.org/doi/10.1145/3386569.3392483`.

Bunny83. SimpleJSON. URL `https://github.com/Bunny83/SimpleJSON`.

M. Chen, S. J. Mountford, and A. Sellen. A study in interactive 3-D rotation using 2-D control devices. In *Proceedings of the 15th annual conference on Computer graphics and interactive techniques - SIGGRAPH '88*, pages 121–129, Not Known, 1988. ACM Press. ISBN 978-0-89791-275-4. doi: 10.1145/54852.378497. URL `http://portal.acm.org/citation.cfm?doid=54852.378497`.

S. Glen. Jaccard Index, 2022. URL `https://www.statisticshowto.com/jaccard-index/#:~:text=The%20Jaccard%20similarity%20index%20(sometimes,more%20similar%20the%20two%20populations.`

A. J. Hanson. THE ROLLING BALL. In *Graphics Gems III (IBM Version)*, pages 51–60. Elsevier, 1992. ISBN 978-0-12-409673-8. doi: 10.1016/B978-0-08-050755-2.50023-3. URL `https://linkinghub.elsevier.com/retrieve/pii/B9780080507552500233`.

K. Hinckley, J. Tullio, R. Pausch, D. Proffitt, and N. Kassell. Usability analysis of 3D rotation techniques. In *Proceedings of the 10th annual ACM symposium on User interface software and technology - UIST '97*, pages 1–10, Banff, Alberta, Canada, 1997. ACM Press. ISBN 978-0-89791-881-7. doi: 10.1145/263407.263408. URL `http://portal.acm.org/citation.cfm?doid=263407.263408`.

A. Kageyama. Keyboard-based control of four-dimensional rotations. *Journal of Visualization*, (19):319–326, Sept. 2005. URL https://link.springer.com/article/10.1007/s12650-015-0313-y#ref-CR4.

A. Kageyama. A visualization method of four-dimensional polytopes by oval display of parallel hyperplane slices. (19):417–422, Nov. 2015. URL https://link.springer.com/article/10.1007%2Fs12650-015-0319-5.

kou yeung. WebGLInput. URL https://github.com/kou-yeung/WebGLInput.

Mathoma. Geometric Algebra in 2D - Two Reflections is a Rotation, Nov. 2016. URL https://www.youtube.com/watch?v=Hy2gbdbrJZ8&list=PLpzmRsG7u_gqaTo_vEseQ7U8KFvtiJY4K&index=7.

Mathoma. Geometric Algebra - 3D Rotations and Rotors, Apr. 2017. URL https://www.youtube.com/watch?v=iwQlrgAduMg&list=PLpzmRsG7u_gqaTo_vEseQ7U8KFvtiJY4K&index=15.

K. Matsumoto, N. Ogawa, H. Inou, S. Kaji, Y. Ishii, and M. Hirose. *Polyvision: 4D Space Manipulation through Multiple Projections*. PhD thesis, The University of Tokyo, Kyoto University, Kyushu University, 2019. URL https://www.math.kyoto-u.ac.jp/~kaji/papers/SIGGRAPH2019_Polyvision_final.pdf.

M. Parker. *Things to make and do in the fourth dimension*. Particular Books, 2014.

I. Quilez. Distance Functions. URL https://www.iquilezles.org/www/articles/distfunctions/distfunctions.htm.

I. Quilez and P. Jeremias Vila. Shadertoy, Feb. 2013. URL https://www.shadertoy.com/.

A. Radiya-Dixit. Visualizing the Fourth Dimension, Apr. 2017. URL https://researchblog.duke.edu/2017/04/26/visualizing-the-fourth-dimension/.

S. Sheppard. TRON Legacy: Oh how far we've come in computer rendering, Dec. 2010. URL https://labs.blogs.com/its_alive_in_the_lab/2010/12/tron-legacy.html.

K. Shoemake. *Arcball Rotation Control*. PhD thesis, University of Pennsylvania, Philadelphia, 1994. URL https://research.cs.wisc.edu/graphics/Courses/559-f2001/Examples/Gl3D/arcball-gems.pdf.

slehar. Clifford Algebra: A visual introduction, Mar. 2014. URL https://slehar.wordpress.com/2014/03/18/clifford-algebra-a-visual-introduction/.

The_Art_of_Code. Ray Marching for Dummies!, Dec. 2018. URL https://www.youtube.com/watch?v=PGtv-dBi2wE.

The_Art_of_Code. Ray Marching Simple Shapes, Jan. 2019a. URL https://www.youtube.com/watch?v=Ff0jJyyiVyw.

The_Art_of_Code. Writing a ray marcher in Unity, Dec. 2019b. URL https://www.youtube.com/watch?v=S8AWd66hoCo.

The_Art_of_Code. How to texture a procedural object, Dec. 2020. URL https://www.youtube.com/watch?v=VaYyPTw0V84.

The_Art_of_Code. Setting up Materials, Apr. 2021. URL https://www.youtube.com/watch?v=Agf188Q8EAc&t=462s.

Z. Tianli. 2D Cross-Section in Unity, May 2018a. URL `https://luicat.github.io/2018/05/19/2D-clipping-in-unity.html`.

Z. Tianli. *4D Rendering: Projection & Cross Section*. PhD thesis, KTH Royal Institute of Technology, May 2018b. URL `https://luicat.github.io/2018/05/23/4D-rendering.html`.

X. Yan, C.-W. Fu, and A. J. Hanson. Multitouching the Fourth Dimension. *Computer*, 45(9): 80–88, Sept. 2012. ISSN 0018-9162. doi: 10.1109/MC.2012.77. URL `http://ieeexplore.ieee.org/document/6165243/`.

J. Zhou. *Visualization of Four Dimensional Sace and Its Applications*. PhD thesis, Purdue University, Dec. 1991. URL `https://docs.lib.purdue.edu/cgi/viewcontent.cgi?article=1921&context=cstech`.

J. Šafránková. The van Hiele Model of Geometric Thinking. pages 72–75, Praha, 2012. Matfyzpress. ISBN 978-80-7378-224-5. URL `https://www.mff.cuni.cz/veda/konference/wds/proc/pdf12/WDS12_112_m8_Vojkuvkova.pdf`.