BACHELOR

Wythoff's construction of regular polytopes

Wennekes, Rick J.

Award date:
2020

Eindhoven University of Technology

Bachelor Final Project

2WH40

Wythoff's construction of regular polytopes

Author
R.J. Wennekes
(1021867)

Supervisor
prof. dr. F.G.M.T.
Cuypers

Assessor
dr. H.J.M. Sterk

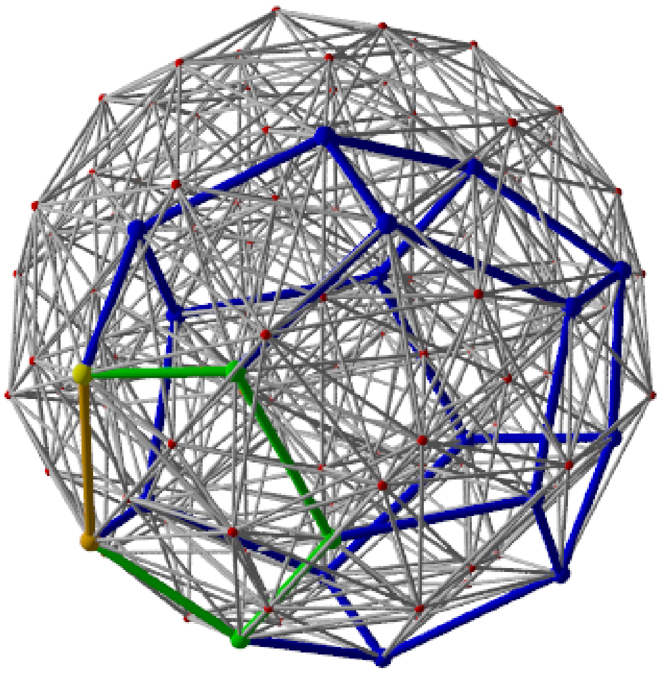


Figure 1: 120 Cell

Eindhoven, December 2020

Abstract

In this paper Wythoff's construction of regular polytopes is investigated and proven. The goal of this paper is to use Wythoff's construction to construct and illustrate 4-dimensional polytopes. All work will be done in the Euclidean space with all its properties. To understand the basics of the construction, we look at the two and three dimensional polytopes and prove the existence of reflections. Furthermore the general case of an n-dimensional polytope is also proven to admit reflections. The properties of these reflections are enlightened, proven and used. We will see that the symmetry group of a regular polytope is generated by some set of reflections. This knowledge gives rise to Wythoff's construction. At last Wythoff's construction is used to construct the regular heptagon, the icosahedron and some illustrations that depict regular 4-dimensional polytopes.

Preface

This Bachelor Final Project was concluded in a difficult period, it started on the 3^{rd} of February 2020 and has been achieved in December 2020. During this period there was a global pandemic outbreak, due to this the university was shut down. For this reason I want to thank my supervisor prof. dr. F.G.M.T. Cuypers for being able to guide me through online meetings and for the help on my thesis. His weekly availability and fast responses on my e-mails were useful for the progress.
Furthermore I want to thank my fellow students and my girlfriend with whom I got to share experiences during this difficult time and thus kept me motivated. I of course also want to thank my parents, I was able to tell them about my struggles. Despite of their mathematical background they did not master my subject but it was helpful to hear their thinking process.

Contents

Chapter 1

Introduction

A triangle, a square, a pyramid and a cube are well known concepts. In fact most people know what they are. These are geometric objects and they are all examples of a polytope. The triangle and square are objects in the plane and are therefore 2-dimensional polytopes. The pyramid and cube are defined in space as we know and thus are 3-dimensional polytopes. In two dimensions the polytopes are called polygons and in three dimensions we talk about polyhedra. These types of geometric objects have been known for many centuries and the theory behind them extends into many directions.
In the 19^{th} century mathematicians began to realize that there might exist more dimensions than the three we know and see. They began to consider higher dimensional polytopes. Lüdwig Schläfli [9] was the first to consider polytopes as analogues of polygons and polyhedra. He described the six convex regular 4-dimensional polytopes. By 1854, Bernhard Riemann [4] had firmly established the geometry of higher dimensions which made the idea of higher dimensional polytopes acceptable. An important milestone was reached in 1948 with H.S.M. Coxeter's book Regular Polytopes [2], he summarized the work that had been done and added his own findings.
In this paper we will mainly follow the findings described in Coxeter's book together with another one of his books Regular Complex Polytopes [3]. In these books, Wythoff's construction is discussed and the different regular polytopes are established. We will see a proof for this construction and which polytopes actually exist. Our goal is to construct and illustrate 4-dimensional polytopes using this Wythoff's construction.

Outline

In chapter 2 we will start by defining the environment and all its properties in which we will be working. Then the concept of a polytope and the 2-and 3-dimensional examples are explained. We define the flags of a polytope which will be used a lot in the later chapters together with symmetry operations such as reflections. The existence of such reflections is proven in chapter 3. Moreover, chapter 3 gives the general definition of a regular polytope and the proof for the existence of reflections that generate the group of automorphisms. In chapter 4 we will discuss the properties admitted by the generating reflections of a polytope. We define the fundamental region, discuss the usage of diagrams and which finite set of options exist for regular n-dimensional polytopes. At the end of chapter 4 we will give a proof for the specific construction of such regular polytopes, called Wythoff's construction. In the 5th chapter Wythoff's construction is explained with the use of examples of a polygon and a polyhedron. Two 4-dimensional polytope representations are discussed in detail in chapter 6. After these examples a table of various diagrams and its corresponding polytope representations is given and we will discuss some relations between these polytopes. At last a glossary, scripts of code for Mathematica and POV-Ray are presented in chapter 7.

Chapter 2

Definitions

We will be working in the n-dimensional Euclidean space. In this chapter we will provide definitions. The Euclidean space will be defined as well as some of the properties of the Euclidean space. In paragraph 2.2 some specific features of vectors are explained. Our main subject, a polytope, is defined together with its two and three dimensional analogues. Some subsets of a polytope are called flags and have their own significance which we will be using later.

2.1 Euclidean space

Let E be a vector space, we state the following definitions for general sets of vectors in E.

Definition 2.1.1. A set of vectors $\{\vec{v}_1, \vec{v}_2, \ldots, \vec{v}_n\}$ is said to be **linearly independent** if and only if the vector equation:

$$\lambda_1\vec{v}_1 + \lambda_2\vec{v}_2 + \ldots + \lambda_n\vec{v}_n = 0 \tag{2.1}$$

only has the trivial solution where $\lambda_i = 0$ for all $i = 1, 2, \ldots, n$.

Definition 2.1.2. A set of vectors $V = \{\vec{v}_1, \ldots, \vec{v}_n\}$ is said to **span** a vector space E if every vector $\vec{u} \in E$ is a linear combination of the vectors from V. i.e., $\vec{u} = x_1\vec{v}_1 + x_2\vec{v}_2 + \ldots + x_n\vec{v}_n$ for some $x_1, ..., x_n \in \mathbb{R}$.

Definition 2.1.3. A set of vectors $V = \{\vec{v}_1, \ldots, \vec{v}_n\}$ is called a **basis** of E if V spans E and its vectors are linearly independent. The vectors $\vec{v}_i$ are called **basis vectors** of E.

The **size** of a basis V is equal to the number of vectors contained in V.
After choosing the basis $V = \{\vec{v}_1, \ldots, \vec{v}_n\}$ for the vector space E, the typical notation for a vector $\vec{x} \in E$ is $\vec{x} = (x_1, x_2, ..., x_n)^\mathsf{T}$ where $\vec{x} = x_1\vec{v}_1 + x_2\vec{v}_2 + \ldots + x_n\vec{v}_n$. Here $x_i \in \mathbb{R}$ and are called the **Cartesian coordinates** of $\vec{x}$.
The **dimension** of a vector space E is fixed and is equal to the size of a basis V of E.

Definition 2.1.4. The real **Euclidean** n-dimensional **space** denoted by $\mathbb{E}^n$ is a real vector space of dimension n with [7] a dot product defined on its elements.
The **dot product** $\langle . | . \rangle$ of two vectors $\vec{x}, \vec{y} \in \mathbb{E}^n$ satisfies the following properties [6]:

- $\langle \vec{x} | \vec{y} \rangle = \langle \vec{y} | \vec{x} \rangle$ for all $\vec{x}, \vec{y} \in \mathbb{E}^n$;
- $\langle (\lambda\vec{x} + \mu\vec{y}) | \vec{z} \rangle = \lambda\langle \vec{x} | \vec{z} \rangle + \mu\langle \vec{y} | \vec{z} \rangle$ for all $\vec{x}, \vec{y}, \vec{z} \in \mathbb{E}^n$ and $\lambda, \mu \in \mathbb{R}$;
- $\langle \vec{x} | \vec{x} \rangle$ is **positive semi-definite**, in that $\langle \vec{x} | \vec{x} \rangle \geq 0$, with equality if and only if $\vec{x} = \vec{0}$.

Corollary 2.1.4.1. There exist a **norm** and a **distance** defined on $\mathbb{E}^n$.

- The norm $|\vec{x}|$ of a vector $\vec{x} \in \mathbb{E}^n$ is defined as $\sqrt{\langle \vec{x} | \vec{x} \rangle}$.
- For two vectors $\vec{r}, \vec{s} \in \mathbb{E}^n$ the norm of $\vec{d} = \vec{r} - \vec{s}$ is called the **distance** from $\vec{r}$ to $\vec{s}$ (and vice versa).

Consider a basis for $\mathbb{E}^n$ and the vectors $\vec{x}, \vec{y} \in \mathbb{E}^n$ with $\vec{x} = (x_1, x_2, ..., x_n)^\mathsf{T}$, $\vec{y} = (y_1, y_2, ..., y_n)^\mathsf{T}$ and $x_i, y_i \in \mathbb{R}$.
The **angle** θ between the vectors $\vec{x}, \vec{y}$ is given by $\theta = cos^{-1}(\frac{\langle \vec{x}|\vec{y}\rangle}{|\vec{x}||\vec{y}|})$ with $0 \leq \theta \leq \pi$.
The vectors $\vec{x}, \vec{y}$ are said to be **orthogonal** if the angle between the vectors is $\frac{\pi}{2}$ and thus if $\langle \vec{x}|\vec{y}\rangle = 0$.
The Euclidean norm $|\vec{x}|$ is defined as the **length** of a vector $\vec{x}$. We say that $\vec{x} \in \mathbb{E}^n$ is a **unit vector** if it has length one: $|\vec{x}|= 1$.
We call a set of vectors $\{\vec{u}_1, \vec{u}_2, ..., \vec{u}_k\} \subset \mathbb{E}^n$ **orthonormal** if each $\vec{u}_i$ is a unit vector and if $\vec{u}_i$ and $\vec{u}_j$ are orthogonal for all $i \neq j$.
If the basis for $\mathbb{E}^n$ is orthonormal then the dot product of the vectors $\vec{x}, \vec{y}$ is evaluated by the formula $\langle \vec{x}|\vec{y}\rangle = x_1y_1 + x_2y_2 + ... + x_ny_n$ and the length of $\vec{x}$ is given by $|\vec{x}|= \sqrt{\langle \vec{x}|\vec{x}\rangle} = \sqrt{\sum_{i=1}^{n} x_i^2}$.

At last, for the ease of understanding we will be using some fixed notations throughout this paper. We will refer to the line-segment from A to B as AB and to the angle at B formed by the sides AB and BC as $\measuredangle ABC$. A triangle formed by the points A, B and C will be denoted as $\triangle ABC$. Sometimes it is easier to talk about a point in space instead of a vector. We will consider a point in $\mathbb{E}^n$ to be the location where the corresponding vector emanating from the origin ends.

2.2 Specifics

Definition 2.2.1. A subset $A \neq \emptyset, A \subseteq \mathbb{E}^n$ is called a **linear subspace** of $\mathbb{E}^n$ if A contains the zero vector $\vec{0}$ and if $\vec{x} + \vec{y} \in A$ and $\lambda\vec{x} \in A$ for all $\vec{x}, \vec{y} \in A, \lambda \in \mathbb{R}$. We say, A is closed under vector addition and scalar multiplication.

Definition 2.2.2. A subset $A \neq \emptyset, A \subseteq \mathbb{E}^n$ is said to be an **affine subspace** of $\mathbb{E}^n$ if and only if there exists a vector $\vec{p} \in \mathbb{E}^n$ and a linear subspace $V \subseteq \mathbb{E}^n$ such that
$A = \{\vec{p} + \vec{v} : \vec{v} \in V\} = \vec{p} + V$. [5]

The dimension of an affine subspace $A = \vec{p} + V$ is equal to the size of a basis for V.
The following are fundamental properties of $\mathbb{E}^n$ [6]:

- an orthonormal set is linearly independent;
- each linear subspace of $\mathbb{E}^n$ has an orthonormal basis;
- any orthonormal set in $\mathbb{E}^n$ can be extended to an orthonormal basis of $\mathbb{E}^n$.

An affine **hyperplane** is an $(n - 1)$-dimensional affine subspace of $\mathbb{E}^n$, for example a line is a 1-dimensional hyperplane in the 2-dimensional space.
A vector that is perpendicular to a hyperplane is called a **normal vector** of that hyperplane. If the normal vector $\vec{n}$ of a hyperplane has length one, so $|\vec{n}|= 1$, then it is called a **unitary normal** of that hyperplane.
A **dihedral angle** is the angle between normal vectors of two hyperplanes. In Cartesian coordinates we describe an affine hyperplane with an equation of the form $a_1x_1 + \ldots + a_nx_n = b$ or $\langle \vec{a}|\vec{x}\rangle = b$ with $\vec{a} \in \mathbb{E}^n$ the normal vector of the hyperplane and $b \in \mathbb{Z}$. Note, if $b = 0$ then the hyperplane contains $\vec{0}$.

An **affine mapping** is a transformation of $\mathbb{E}^n$ that preserves collinearity (i.e., all points lying on a line initially still lie on a line after transformation) and ratios of distances (e.g., the midpoint of a line segment remains the midpoint after transformation).

Definition 2.2.3. An **isometry** of $\mathbb{E}^n$ is an affine mapping $\sigma : \mathbb{E}^n \to \mathbb{E}^n$ which preserves distances, i.e. $|\sigma(\vec{x}) - \sigma(\vec{y})|= |\vec{x} - \vec{y}|$ for all $\vec{x}, \vec{y} \in \mathbb{E}^n$.

Lemma 2.2.4. An isometry of the space $\mathbb{E}^n$ also preserves angles.

Proof. Let $\sigma : \mathbb{E}^n \to \mathbb{E}^n$ be an isometry of $\mathbb{E}^n$ and let $A, B, C \in \mathbb{E}^n$ be points in the space. Consider the line segments AB, BC and AC and let c, a and b respectively be the lengths of those line segments.
If α is the angle between AB and AC then the law of cosines states:

$$\alpha = \cos^{-1}(\frac{b^2 + c^2 - a^2}{2 \cdot b \cdot c}).$$

Now consider the points $\sigma(A), \sigma(B), \sigma(C)$ which are the images of A, B, C under σ. Consider $\sigma(a), \sigma(b)$ and $\sigma(c)$ to be the lengths of the line segments $\sigma(B)\sigma(C), \sigma(A)\sigma(C)$ and $\sigma(A)\sigma(B)$. Let $\sigma(\alpha)$ be the angle between $\sigma(A)\sigma(B)$ and $\sigma(B)\sigma(C)$ then the law of cosines states:

$$\sigma(\alpha) = \cos^{-1}(\frac{\sigma(b)^2 + \sigma(c)^2 - \sigma(a)^2}{2 \cdot \sigma(b) \cdot \sigma(c)}).$$

Since σ preserves distances we have that $\sigma(a) = a, \sigma(b) = b$ and $\sigma(c) = c$. We therefore find

$$\sigma(\alpha) = \cos^{-1}(\frac{b^2 + c^2 - a^2}{2 \cdot b \cdot c}) = \alpha.$$

Thus σ also preserves angles. $\square$

Two sets of vectors $V_1, V_2 \subseteq \mathbb{E}^n$ are said to be **congruent** if there exists an isometry σ of $\mathbb{E}^n$ such that $\sigma(V_1) = V_2$.
The **translation** defined by $T_{\vec{v}} : \mathbb{E}^n \to \mathbb{E}^n$ by a vector $\vec{v}$ is the affine mapping $T_{\vec{v}}(\vec{w}) = \vec{w} + \vec{v}$ for all $\vec{w} \in \mathbb{E}^n$. For any pair of vectors $\vec{w}, \vec{u} \in \mathbb{E}^n$ the difference vector $\vec{d}$ after translation is then defined as

$$\vec{d} = T_{\vec{v}}(\vec{w}) - T_{\vec{v}}(\vec{u}) = (\vec{w} + \vec{v}) - (\vec{u} + \vec{v}) = \vec{w} - \vec{u} + \vec{v} - \vec{v} = \vec{w} - \vec{u}. \quad (2.2)$$

So the difference vector remains unchanged under the translation and thus also the distance between the vectors $\vec{w}$ and $\vec{u}$ is preserved by the translation. For this reason, a translation is an isometry.

2.3 Polytope

The main focus of this report is on polytopes. To explain the concept of polytopes we first state some preliminaries.
Let $(\mathcal{P}, \leq)$ be a **poset** (partially ordered set), $\mathcal{P}$ is a set with a transitive anti-symmetric binary relation $\leq$. The word 'partially' indicates that not every pair of elements of a poset needs to be comparable using the binary relation. That is, there may be pairs of elements for which neither comes before the other in the ordering of the poset.
If $A, B, C \in \mathcal{P}$ with $A \leq B \leq C$ then $A \leq C$, and if $A \leq B \leq A$ then $A = B$. If $A \leq B$ or vice versa then A and B are called **incident**. A poset is **connected** if given any $A, B \in \mathcal{P}$, there is a sequence $A = C_0, C_1, \ldots, C_k = B$ in $\mathcal{P}$ for some $k \in \mathbb{N}$, such that C_i and C_{i+1} are incident for $i = 0, \ldots, k-1$. We write $A < B$ if $A \leq B$ but $A \neq B$. If $A < B$, but there is no C such that $A < C < B$ then B is said to **cover** A.

An i-**cell** is a finite subset of an $(i-1)$-dimensional affine subspace of $\mathbb{E}^n$, which is not contained in an $(i-2)$-dimensional affine subspace of $\mathbb{E}^n$.
If an i-cell is a subset of a j-cell then these cells are called **incident cells**, note that $i \leq j$ since a higher dimensional subset cannot be a subspace of a lower dimensional subset.

2.3.1 Polygon

A **polygon** in $\mathbb{E}^2$ is a poset of 1-and 2-cells with the binary relation $\subseteq$. Each 1-cell is called a **vertex** (plural: **Vertices**) and each 2-cell is called an **edge**. Every edge connects two vertices and every vertex is incident to exactly two different edges. Two vertices that share an edge are called **adjacent vertices**. Moreover two adjacent vertices share exactly one edge. By this argument a polygon is not defined on 2 vertices.
We assume that all vertices are connected to each other through some sequence of consecutive adjacent vertices. So two vertices A and B are **connected vertices** if there exists a sequence of vertices $(A = v_0, v_1, v_2, ..., v_k = B)$, for some natural number k, such that the vertices of the pairs (v_i, v_{i+1}) are adjacent vertices.
Then also the edges are connected.

Lemma 2.3.1. A polygon consists of an equal number of edges and vertices.

We denote a polygon consisting of p edges and p vertices, $p \geq 3$, as a p-**gon**.
For example a 4-gon has 4 edges and 4 vertices.
The **length** of an edge e is the distance between the two vertices incident to e.
If all edges are of the same length, the polygon is called **equilateral** [3]. For instance, a rhombus is an equilateral polygon. If all the angles of a polygon are equally valued then the polygon is called **equiangular** [3], an example is a rectangle. A square is both equilateral and equiangular.

2.3.2 Polyhedron

A **polyhedron** in $\mathbb{E}^3$ is a poset of 1-,2-and 3-cells with the binary relation $\subseteq$. Each 3-cell of this set is a polygon, the 2-cells are edges of these polygons and the 1-cells are vertices of the polygons.
The polygons are also called **faces** of the polyhedron.
Each edge of a polyhedron is incident to exactly two faces and two faces share at most one edge. If two faces share an edge then they are called **adjacent faces**.
We assume that all faces are connected to each other through some sequence of consecutive adjacent faces.
An example of a polyhedron is the well-known cube, which has squares as its faces and three squares meet at each vertex.

2.3.3 Flags

Definition 2.3.2. A **flag** in $\mathbb{E}^n$ is a sequence of i-cells that are all mutually incident. It contains at most one i-cell for every $i \in \{1, 2, ..., n\}$.

We will call a flag of n elements a **maximal flag**, whereas a **sub-maximal flag** is a maximal flag that does not contain an n-cell.
Two flags are said to be **adjacent flags** when they have an equal number of elements and differ for exactly one i-cell.

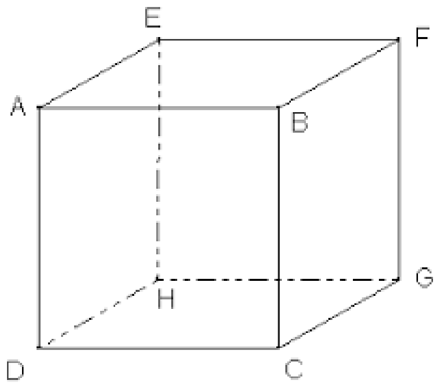


Figure 2.1: A cube with labeled vertices

For the cube in figure 2.1 the vertex A is a 1-cell, the edge AB containing vertex A is a 2-cell and the face $ABCD$ containing the edge AB is a 3-cell. We denote a flag by the sequence of its elements, for this instance $(A, AB, ABCD)$ is a flag. More precisely, $(A, AB, ABCD)$ is a maximal flag of this cube. The sub-maximal flag (A, AB) is incident to the maximal flags $(A, AB, ABCD)$ and $(A, AB, ABFE)$.
The flags $(A, AB, ABCD)$ and $(A, AB, ABFE)$ are adjacent.

2.3.4 Polytope

We have seen the definitions of a polygon and a polyhedron and will now discuss the following definition for a general polytope.

Definition 2.3.3. A **polytope** Π in $\mathbb{E}^n$ is a poset of i-cells with $i \in \{1, ..., n\}$ and containing the empty set $\emptyset$ and the polytope Π itself (which we call the 0- and $(n+1)$-cell respectively) with the binary relation $\subseteq$. Every sequence of i-cells of Π is contained in a maximal sequence of length $n+2$ containing one i-cell for each $i \in \{0, 1, ..., n+1\}$.
The elements of Π have to satisfy [11]:

Property 2.3.4. (Diamond property) Given a $(k-1)$-cell C_{k-1} and a $(k+1)$-cell C_{k+1} of a polytope for which $C_{k-1} \subseteq C_{k+1}$, then there exist exactly two k-cells C_k, C'_k such that $C_{k-1} \subseteq C_k, C'_k \subseteq C_{k+1}$.

For every flag F of Π that does not contain an i-and $(i+1)$-cell, the maximal flags of Π containing F form a polygon and are thus connected. By the diamond property any of these maximal flags is adjacent to exactly two other maximal flags, which implies:

Property 2.3.5. (Flag connectedness) Let F, F' be two flags of a polytope. For any pair of maximal flags M, M' where M contains F and M' contains F', there exists a sequence of maximal flags $(M = M_1, M_2, ..., M_k = M')$ such that M_i is adjacent to M_{i+1} for $i = 1, ..., k-1$.

An element of Π that is covered by Π is also called a **cell**.

2.4 Symmetry

A **symmetry** of a polytope Π contained in $\mathbb{E}^n$ is an isometry of $\mathbb{E}^n$ which maps each vertex to a vertex, edge to an edge, etc and it preserves the incidence between these elements, i.e. $\sigma : \Pi \to \Pi$ which means that σ maps the elements of the polytope to other elements of the polytope such that the image is again the polytope Π. The simplest symmetry example is the **identity** I, which leaves every element invariant. If a symmetry σ is applied after a symmetry τ we will denote this by $\sigma \circ \tau$. It holds that $\sigma \circ \tau$ is again a symmetry, this is a property of isometries. Namely if τ and σ are both bijective mappings that preserve distances then $\sigma \circ \tau$ is also bijective and distance preserving and is therefore also an isometry. Now if it holds that $\tau(\Pi) = \Pi$ and $\sigma(\Pi) = \Pi$ then it also holds that $\tau \circ \sigma(\Pi) = \Pi$, thus $\tau \circ \sigma$ is also a symmetry.
The order of a symmetry σ is the smallest $k \in \mathbb{N}_{>0}$ such that $\sigma^k = \sigma \circ \sigma ... \circ \sigma = I$.

Definition 2.4.1. Let G be a set and $*$ a binary operation on G. A tuple $(G, * : G \times G \to G, inv : G \to G, e \in G)$ is called a **group** if:

- It is associative, i.e. for all $f, g, h \in G$ it holds $(f * g) * h = f * (g * h)$.
- e is an identity element for $*$.
- For each $g \in G$, g^{-1} satisfies $g * g^{-1} = e = g^{-1} * g$.

These properties are called the **group axioms**. By this definition, if Π is a polytope then the set $Sym(\Pi)$ of all possible symmetries of Π is a group since we have:

- For all $\sigma, \tau, \rho \in Sym(\Pi)$ it holds $(\sigma \circ \tau) \circ \rho = \sigma \circ (\tau \circ \rho)$.
- I is an element of $\in Sym(\Pi)$.
- For every $\sigma \in Sym(\Pi)$ there exists $\sigma^{-1} \in Sym(\Pi)$ such that $\sigma^{-1} \circ \sigma = I = \sigma \circ \sigma^{-1}$.
- If $\sigma, \tau \in Sym(\Pi)$ then also $\sigma \circ \tau \in Sym(\Pi)$.

The set $Sym(\Pi)$ is called the **symmetry group** of a polytope Π.

2.4.1 Reflection

The **axis** of an isometry σ of the space $\mathbb{E}^n$ is

$$axis(\sigma) := \{\vec{x} \in \mathbb{E}^n : \sigma(\vec{x}) = \vec{x}\};$$

it is the set of vectors left invariant by the isometry.

Definition 2.4.2. A **reflection** is a symmetry of the Euclidean space that leaves invariant every vector on an affine hyperplane. The affine hyperplane is called the reflection axis or mirror. The reflection interchanges the two half-spaces into which the reflection axis decomposes the whole space, such that the line-segment between any vector of the space and its image after reflection is perpendicularly bisected by the reflection axis.

The **inverse** R^{-1} of a transformation R reverses the effect of R, such that $RR^{-1} = I = R^{-1}R$ where I is the identity.
If R is a reflection then R is its own inverse, $R = R^{-1}$, and we have: $R^2 = I$. Because of this, we say that a reflection is of order 2. i.e., reflecting twice in the same reflection axis leaves everything invariant.

Now that we have a clear understanding of what a reflection is, we can ask ourselves the following question. How does one find or compute the reflection image of a vector? Suppose we have the reflection R in the hyperplane V, with the unitary normal $\vec{n}$. To determine the reflection image in V of a vector $\vec{v}$ we distinguish two cases: either the hyperplane goes through the origin $\vec{0}$ or it does not and then there is a vector $\vec{u} \neq \vec{0}$ such that $\vec{u}$ is contained in the hyperplane.

> **Lemma 2.4.3.** If the hyperplane V contains $\vec{0}$ then the reflection image of a vector $\vec{v}$ in the hyperplane V is defined by:
>
> $$R(\vec{v}) = \vec{v} - 2\langle \vec{v}|\vec{n}\rangle \vec{n}. \tag{2.3}$$
>
> where $\vec{n}$ is a unitary normal of V.

Proof. Let $\vec{x}$ be a vector and let $\vec{n}$ be the unitary normal of the plane V where V goes through the origin. Then V is defined by $\langle \vec{x}|\vec{n}\rangle = 0$. To find the reflection image in V of a vector $\vec{v}$ we need to consider the line through $\vec{v}$ and perpendicular to the plane. Let l be this line. Then l is defined by $\vec{x} = \vec{v} + \lambda\vec{n}$. For the intersection of l with V we find: there is a $\hat{\lambda}$ such that $\langle(\vec{v} + \hat{\lambda}\vec{n})|\vec{n}\rangle = 0$. Furthermore, $\langle \vec{n}|\vec{n}\rangle = |\vec{n}|= 1$ and thus we have $\hat{\lambda} = -\langle \vec{v}|\vec{n}\rangle$. Now $-\langle \vec{v}|\vec{n}\rangle\vec{n}$ is the distance vector from $\vec{v}$ to V and $\vec{u} = \vec{v} + -(\langle \vec{v}|\vec{n}\rangle)\vec{n} \in V$.
Let R be the reflection in V. The reflection image $R(\vec{v})$ of $\vec{v}$ is also on line l, has the same distance to V as $\vec{v}$ and therefore we find $R(\vec{v}) = \vec{v} - 2\langle \vec{v}|\vec{n}\rangle\vec{n}$.
Note: if $\vec{v}$ is a vector in the plane V then $\langle \vec{v}|\vec{n}\rangle = 0$ and thus $R(\vec{v}) = \vec{v}$. □

However if V does not go through $\vec{0}$ but through a vector $\vec{u}$, we can apply the translation by $-\vec{u}$ to the hyperplane and the vector $\vec{v}$ such that the resulting hyperplane V' is parallel to V and does go through the origin. After translation we use the equation 2.3 to reflect in V' and after reflection we translate by the vector $+\vec{u}$.
Let R' be the reflection in the hyperplane V'. For the reflection image of the shifted vector $\vec{v} - \vec{u}$ by R' equation 2.3 holds. So for the reflection image of $\vec{v}$ in V it holds that we then have $R(\vec{v}) = R'(\vec{v} - \vec{u}) + \vec{u} = \vec{v} - \vec{u} - 2\langle(\vec{v} - \vec{u})|\vec{n}\rangle\vec{n} + \vec{u}$. Thus we find:

> **Lemma 2.4.4.** If the hyperplane V does not go through $\vec{0}$ but through $\vec{u}$ then the reflection image of a vector $\vec{v}$ in the hyperplane V is defined by:
>
> $$R(\vec{v}) = \vec{v} - 2\langle \vec{v}|\vec{n}\rangle\vec{n} + 2\langle \vec{u}|\vec{n}\rangle\vec{n} \tag{2.4}$$
>
> where $\vec{n}$ is a unitary normal of V.

Chapter 3

Symmetries of polytopes

After discussing the main concepts of this paper and the definitions needed, we can now get more into the details of polytopes. In this chapter the existence of symmetries, more precisely reflections, for regular polytopes will be proven. We start with polygons, here the practicalities are easier to understand. Then we will introduce a new definition and some properties on the polytopes in higher dimensions and prove that also for arbitrary n there exist reflections that are symmetries of regular polytopes in n-dimensional space.

3.1 2-dimensional

To understand the various symmetries of polytopes we first take a look at the simplest case, regular polygons in the Euclidean plane.

Definition 3.1.1. A **regular polygon** is a polygon that is both equilateral and equiangular.

An example of a regular polygon is a square. Throughout this paper we will mainly consider regular polygons for as they have some nice properties which will become clear in chapter 4.

Theorem 3.1.2. Every regular polygon on $n \geq 3$ vertices admits reflections as symmetries.

More precisely, we will prove that a regular polygon admits reflections in the bisector of any angle between three consecutively adjacent vertices of the polygon and in the perpendicular bisector of any two adjacent vertices.
To prove the existence of symmetries for regular polygons we will make use of the following lemma.

Lemma 3.1.3. For every regular polygon there exists a point with an equal distance to all vertices of the polygon.

Proof. Let there be a regular polygon on $n \geq 3$ vertices, with its vertices numbered from 1 to n such that consecutive adjacent vertices are labeled with consecutive numbers modulo n. Considering the perpendicular bisectors of vertex pairs $(1,2)$ and $(2,3)$ we find one point called M where these lines intersect. These perpendicular bisectors cannot be parallel to each other since the angle $\measuredangle 123$ is not 180 degrees. If $\measuredangle 123$ would be 180 degrees then the regular polygon would be a straight line since it is equiangular, this is in contradiction to a polygon being finite. For M we know that $d(1,M) = d(2,M) = d(3,M)$. By definition of a regular polygon we also know that $d(1,2) = d(2,3) = d(3,4)$. Thus the triangles $\triangle 1M2$ and $\triangle 2M3$ are congruent to one another and are isosceles triangles. They therefore both have two equal base angles. So in total there are four equally valued base angles with value α.
Again by the definition of a regular polygon we have that $\measuredangle n12 = \measuredangle 123 = \measuredangle 234$. The angle $\measuredangle 123$ consists of two equal base angles $= 2\alpha = \measuredangle 234$ where $\measuredangle 23M = \alpha$ and therefore $\measuredangle M34 = \alpha$.
It follows now that $\triangle 3M4$ is also congruent to $\triangle 2M3$ since they have two sides and the angle in between these sides equal.
From this we can conclude that vertex 4 also has the same distance to the point M. Changing the triangles and angles in this proof by adding 1 to all used vertex numbers we are now able to analogously prove that vertex 5 also has the same distance to point M. In this way we prove via induction that every vertex $1 \leq i \leq n$ has the same distance to point M. We have proven that there

exists a point with an equal distance to all vertices of a regular polygon and it is called the **midpoint** M. ☐

To continue the proof of Theorem 3.1.2: We will discuss two cases. One where we prove proposition 3.1.4 and a second where we prove proposition 3.1.5.

Proposition 3.1.4. The bisector of any angle between three consecutive vertices of the polygon is the axis of a reflection for a regular polygon.

Proof. Consider a regular polygon on n vertices where $n \geq 3$. Without loss of generality we may assume that we can number the vertices with the numbers 1 to n doing so counter clock-wise modulo n. Let A be the vertex with number 1, B the vertex with number n and B' the vertex with number 2. B and B' are now the neighbors of A. Let line k be the bisector of $\measuredangle BAB'$. By definition of the polygon we know that $BA = AB'$ and by lemma 3.1.3 there exists a point M such that $AM = BM = B'M$. M is the midpoint of the polygon and lies on line k. Now we know that $\triangle BMA$ is congruent to $\triangle AMB'$ having all three pairs of sides mutually equal.
And therefore B and B' are each other's mirror-image in line k.
Using induction we prove that for every subsequent neighbors C and C' respectively of the points B and B' it holds that C and C' are each other's mirror-image. Figure 3.1 represents such an instance.
Base case: We have that vertices n and 2 are each other's mirror-image in line k.
Induction hypothesis: Suppose that for a vertex $(i) \geq 1$ we have that $(n+2-i)$ is its mirror-image in line k.
Induction step: Consider vertices $(i+1)$ and $(n+1-i)$. Again by definition of the regular polygon we have $(i)(i+1) = (n+2-i)(n+1-i)$ and by definition of the midpoint M we have $(i)M = (i+1)M = (n+2-i)M = (n+1-i)M$. Thus $\triangle(i)M(i+1)$ is congruent to $\triangle(n+2-i)M(n+1-i)$. And therefore $(i+1)$ and $(n+2-i)$ are each other's mirror-image in line k.
Moreover, there exists a reflection axis for the polygon on the bisector of the angle between any three consecutive vertices. ☐

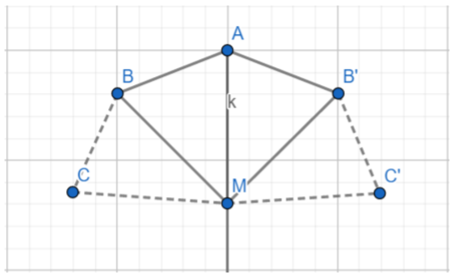


Figure 3.1: Representation of case 1

Proposition 3.1.5. The perpendicular bisector of any two adjacent vertices is the axis of a reflection for a regular polygon.

Proof. Consider a regular polygon on $n \geq 3$ vertices. Without loss of generality we may assume that we can number the vertices with the numbers 1 to n doing so counter clock-wise modulo n. Let A be the vertex with number n, B the vertex with number 1. Consider the perpendicular bisector l of the two adjacent vertices A and B. Line l obviously intersects the midpoint M. Let C be

the second neighbor of A and C' the second neighbor of B. We know by definition of the polygon that $AC = BC'$ and by definition of the midpoint that $AM = BM = CM = C'M$. Leading to the conclusion that $\triangle AMC$ is congruent to $\triangle BMC'$ and therefore C and C' are each others mirror-image in line l. Via induction we will prove that in every subsequent pair of second neighbors the vertices are also each other's mirror image in line l. Figure 3.2 represents such an instance.
Base case: We have that vertices $n-1$ and 2 are each other's mirror-image in line l.
Induction hypothesis: Suppose that for a vertex $(i) \geq 1$ we have that $(n+1-i)$ is its mirror-image in line l.
Induction step: Consider vertices $(i+1)$ and $(n-i)$. By definition of the regular polygon we have $(i)(i+1) = (n+1-i)(n-i)$ and by definition of the midpoint M we have $(i)M = (i+1)M = (n+1-i)M = (n-i)M$. Thus $\triangle(i)M(i+1)$ is congruent to $\triangle(n+1-i)M(n-i)$. And therefore $(i+1)$ and $(n-i)$ are each other's mirror-image in line l.
Furthermore, there exists an axis of reflection for the polygon on the perpendicular bisector of any two adjacent vertices. □

This concludes the proof of Theorem 3.1.2. Every regular polygon on $n \geq 3$ vertices admits reflections. More specifically, it admits reflection in the bisector of any angle between three consecutive vertices of the polygon and in the perpendicular bisector of any two adjacent vertices.

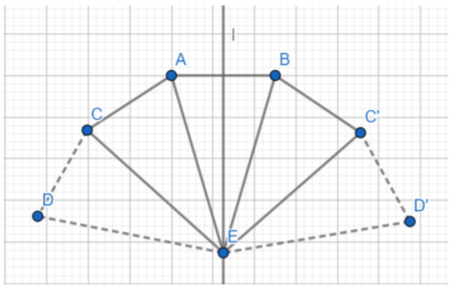


Figure 3.2: Representation of case 2

3.2 3-dimensional

A set of symmetries S of a polytope Π is **transitive** on the maximal flags of Π if for each pair of maximal flags (F, G) of Π there exists a $\sigma \in S$ such that $\sigma(F) = G$.
We have discussed what it means for a polygon to be regular. From now on we will use the general definition for a regular polytope, which is as follows.

Definition 3.2.1. A polytope is said to be a **regular polytope** if and only if its symmetry group is transitive on its maximal flags.

Lemma 3.2.2. The i-cells of a regular n-dimensional polytope are regular $i-1$-dimensional polytopes.

Proof. If Π is a regular n-dimensional polytope, then its symmetry group $Sym(\Pi)$ is transitive on its maximal flags. Let C be an i-cell of Π and let $F = (C_1, ..., C_{i-1}, C, C_{i+1}, ..., C_n)$ be a maximal flag of Π. Let F_1, F_2 be maximal flags of C then their elements are elements of C. They can be extended by combining both with the sequence of elements $C_{i+1}, ..., C_n$, let us denote these combinations as $F_1' = (F_1, C_{i+1}, ..., C_n)$ and $F_2' = (F_2, C_{i+1}, ..., C_n)$.

F_1', F_2' are maximal flags of Π. The symmetry group of Π is transitive on its maximal flags, so there exists a symmetry $\sigma \in Sym(\Pi)$ such that $\sigma(F_1') = F_2'$. Since the last $n-i$ elements of F_1' and F_2' are equal it follows that $\sigma(F_1) = F_2$. Since this holds for arbitrary maximal flags of C we have that $Sym(\Pi)$ is also transitive on the maximal flags of C. We find that the symmetry group of C is transitive on its maximal flags and thus C is an $i-1$-dimensional regular polytope. □

As we have developed some handles to understand the symmetries in regular polygons we can expand our knowledge to understand more about symmetry in regular polyhedra.

3.2.1 Midpoint

First we need to establish that for every regular polyhedron there exists a unique midpoint.

Lemma 3.2.3. For each regular polyhedron there exists a unique point with an equal distance to all vertices of the polyhedron.

Proof. Consider the two vertices A and B of one face of a regular polyhedron. Because all faces are congruent we can choose any of the faces of the polyhedron, let us use the face F. Let the **perpendicular plane** of A and B be the plane that contains all points in $\mathbb{E}^n$ that have an equal distance to A and B.
Consider the perpendicular plane X of the vertices A and B. F is a polygon and thus contains at least three vertices. Let C be a vertex of F unequal to A and B. Consider the perpendicular plane Y of the vertices A and C. The two planes X, Y intersect the face F in the point N. According to lemma 3.1.3 N is the midpoint of F.
Let D be an adjacent vertex of A that is not a vertex of F. D is a vertex of an adjacent face of F. Consider the perpendicular plane Z of the vertices A and D. Let M be the point of intersection of the planes X, Y and Z. Since X and Y are perpendicular to F, their intersection is also perpendicular to F. From this we know that MN is perpendicular to F.
For all vertices V, W of F consider the triangles $\triangle VNM$ and $\triangle WNM$, it holds that $VN = WN$, $\measuredangle VNM = \measuredangle WNM = 90°$ and $NM = NM$ so $\triangle VNM$ and $\triangle WNM$ are congruent. Furthermore it follows that $VM = WM$, so M has the same distance to all vertices of F.

Claim 3.2.4. This unique point M satisfies lemma 3.2.3.

In order to prove this claim we must show that M is always the same point for any chosen vertices. This follows since by definition the polyhedron is connected on its faces. Then there exists a sequence of faces from F to any other face of the polyhedron such that every subsequent pair of faces share an edge.
F will have an edge in common with the next face, say S where S is the adjacent face of F containing vertex D, and thus the point M has the same distance to both vertices at this edge and we can then prove that M is at the same distance from all other vertices of both faces F and S:

Namely, following the aforementioned reasoning: using the perpendicular planes X and Z we find a point N' which is the midpoint of the face S and see that $N'M$ is perpendicular to the face S since M is on the intersection of the two perpendicular planes. We can now consider the triangles $\triangle ANM$ and $\triangle AN'M$. Both triangles contain the side AM. The sides AN and AN' are of equal length since the faces F and S are congruent polygons with midpoints N and N' and A is a vertex of both polygons. Furthermore, both triangles are right-angled with their hypotenuse being AM. So the two triangles are congruent and thus we know that NM and $N'M$ are of equal length.

Thus if two faces F and S are adjacent then the point M with an equal distance to the vertices of F is also the same distance from the vertices of S. It follows that if there is a sequence of consecutive adjacent faces between the faces F and G of a regular polyhedron then the unique point M with an equal distance to the vertices of F has the same distance to the vertices of the face G.
By definition of the polyhedron we know that such a sequence exists.
So we can now conclude that M has an equal distance to the vertices of every face. We call this point M the midpoint of the polyhedron. □

3.2.2 Symmetry of polyhedron

As a regular polygon admits reflections we will now prove that a regular polyhedron admits reflections as well. We will do so using the elements of a polyhedron and the flags in which they are contained. We need the following proposition:

Proposition 3.2.5. If for a regular polytope there exists a unique midpoint then this midpoint can not be contained in the affine span of any of the cells of the polytope.

Proof. Let $F = (C_1, ..., C_{n-1}, C)$ be a maximal flag of a regular polytope Π. We will prove by induction on the elements of F that the midpoint O of Π can not be contained in the span of any element of F.
Base case: Suppose O is contained in the affine span of a cell C of Π. Consider the maximal flag $G = (C_1, ..., C_{n-1}, D)$, G is an adjacent flag of F so there exists a symmetry $\sigma \in Sym(\Pi)$ such that $\sigma(F) = G$. Then there must exist a point O' in the affine span of D with $\sigma(O) = O'$. By definition we have that O is left invariant by every symmetry of the polytope so $O' = O$, thus O is contained in the affine span of C_{n-1}.
Induction hypothesis: Suppose that O is contained in the affine span of an i-cell C_i of Π where C_i is an element of $F_1 = (C_1, ..., C_i, D_{i+1}, ..., D_{n-1}, D)$ for some $i = 2, ..., n-1$.
Induction step: Consider the maximal flag $F_2 = (C_1, ..., C_{i-1}, D_i, ..., D_{n-1}, D)$, F_2 is an adjacent flag of F_1 so there exists a symmetry $\sigma \in Sym(\Pi)$ such that $\sigma(F_1) = F_2$. Then there must exist a point O'' in the affine span of D_i with $\sigma(O) = O''$. By definition we have that O is left invariant by every symmetry of the polytope so $O'' = O$, thus O is contained in the affine span of C_{i-1}.

It now follows that if O is in the affine span of C then $O = C_1$, so O is a vertex of the polytope. However for all vertices it also holds that there exists a symmetry $\sigma \in Sym(\Pi)$ that map the vertex onto another vertex, but O is left invariant by every symmetry of Π.
Therefore we know that O can not be contained in the affine span of any of the cells of Π. □

With this we will now prove the following theorem.

Theorem 3.2.6. Every regular polyhedron in $\mathbb{E}^n$ admits reflection symmetries.

Proof. Let P be a regular polyhedron. For any maximal flag $(C, BC, BCD...)$, where $BCD...$ depicts a face of the polyhedron on the vertices B, C, D and some yet to be determined vertex/vertices, we find some adjacent flags $(B, BC, BCD...), (C, CD, BCD...), (C, BC, ...ABC)$.
The symmetry group of P is transitive on its maximal flags. So these three flags are each the result of applying some symmetry $\sigma_i \in Sym(P)$ to $(C, BC, BCD...)$.
Suppose σ_1 is the symmetry such that $\sigma_1(C, BC, BCD...) = (B, BC, BCD...)$, then σ_1 leaves the edge BC and the face $BCD\ldots$ invariant. As a result their midpoints, O and N respectively, are also left invariant by σ_1. These two midpoints span a line in the bounding hyperplane containing the face $BCD\ldots$. The midpoint M of the polyhedron is also left invariant since σ_1 is a symmetry of

the polyhedron.
Because of proposition 3.2.5 we know that M is not contained in its bounding hyperplane and therefore we know that the points O, N, M span a hyperplane, these points are all left invariant and thus the hyperplane is left invariant. For this reason σ_1 is the reflection in the hyperplane $\rho_1 = ONM$.
Suppose σ_2 is the symmetry such that $\sigma_2(C, BC, BCD...) = (C, CD, BCD...)$, then σ_2 leaves the vertex C and the face $BCD\ldots$ invariant. Analogue to the proof for σ_1 being a reflection we find that σ_2 leaves the hyperplane spanned by the three points C, N, M invariant and is therefore the reflection in the hyperplane $\rho_2 = CNM$.
Suppose σ_3 is the symmetry such that $\sigma_3(C, BC, BCD...) = (C, BC, ...ABC)$, then σ_3 leaves the vertex C and the edge BC invariant. Again analogue to the proof for σ_1 being a reflection we find that σ_3 leaves the hyperplane spanned by the three points C, O, M invariant and is therefore the reflection in the hyperplane $\rho_3 = COM$.
Thus every symmetry $\sigma \in Sym(\Pi)$ that transforms a maximal flag into any adjacent maximal flag is a reflection. □

3.3 n-dimensional

We have proven that each regular polygon or polyhedron has a unique point with an equal distance to every vertex of that polygon or polyhedron. This is the midpoint. To prove the existence of a midpoint for higher dimensional polytopes we will use the following lemmas.

Lemma 3.3.1. The number of symmetries of a regular polytope Π is finite.

Proof. Consider a set of points P containing the minimal number of points of a polytope Π contained in $\mathbb{E}^n$ such that P spans the space $\mathbb{E}^n$. By the definition of regularity, applying the symmetry group $Sym(\Pi)$ to the set P results in all points of Π.
Suppose that the number of symmetries in $Sym(\Pi)$ is infinite, then there always exists some symmetry $\sigma \in Sym(\Pi)$ such that σ is a mapping defined on the points of Π. Since we already have all points of Π we know that $\sigma(P)$ can not result in any new points of the polytope and therefore σ leaves Π invariant. The points of Π span $\mathbb{E}^n$ thus σ leaves the space invariant and is therefore the identity symmetry.
Thus the number of symmetries in $Sym(\Pi)$ is finite. □

Lemma 3.3.2. Let Π be a regular polytope. For any pair of i-cells A, B of Π there exists a symmetry $\tau \in Sym(\Pi)$ such that $\tau(A) = B$.

Proof. Let Π be a regular polytope and let A, B be two i-cells of Π. Every i-cell is contained in some maximal flag of Π. So for A and B there exist maximal flags F, F' of Π such that A is an element of F and B is an element of F'. Since the symmetry group of Π is transitive on its maximal flags, there exists a symmetry $\tau \in Sym(\Pi)$ such that $\tau(F) = F'$ and thus we have $\tau(A) = B$. □

Using these results we can use the symmetry operations between vertices to prove the following.

Theorem 3.3.3. For every regular polytope there exists a point with an equal distance to all vertices of the polytope.

Proof. Let Π be a regular polytope with A a random point of Π. Let $\sigma_1, \sigma_2, ..., \sigma_k$ be all symmetries in $Sym(\Pi)$, by lemma 3.3.1 the number of symmetries is finite. If $O = \sigma_1(A) + \sigma_2(A) + ... + \sigma_k(A)$

then O is left invariant by every $\sigma \in Sym(\Pi)$ since $\sigma \circ \sigma_i$ is again a symmetry of the polytope and thus equal to σ_j for some $j \in \{1, ..., k\}$ uniquely depending on i. We have

$$\sigma(O) = \sigma \circ \sigma_1(A) + \sigma \circ \sigma_2(A) + ... + \sigma \circ \sigma_k(A) = \sigma_1(A) + \sigma_2(A) + ... + \sigma_k(A) = O$$

So indeed O is left invariant. The symmetries preserve incidence and distances. So the distance from O to a vertex A of the polytope is preserved. Say $d(O, A) = a$. If we have $\tau(A) = B$ for some $\tau \in Sym(\Pi)$ then $a = d(\tau(O), \tau(A)) = d(O, B)$ since O is left invariant by τ.
Because of lemma 3.3.2 there always exists such a symmetry mapping A to B and therefore O has an equal distance to all vertices of the polytope and is thus the midpoint of Π. □

Recall proposition 3.2.5. The knowledge of the existence of a midpoint in a regular polytope and whether or not it lies in one of the cells will come in handy for the following. We assume that our polytopes of interest are regular and thus we already know that there exists a symmetry mapping a flag to an adjacent flag. However, what type of symmetry is this?

Theorem 3.3.4. Let Π be a regular polytope and let F_1, F_2 be two adjacent maximal flags of Π. If $\sigma \in Sym(\Pi)$ is the symmetry such that $\sigma(F_1) = F_2$ then σ is a reflection.

Proof. Let Π be a regular polytope in $\mathbb{E}^n$. Let C_i and C_{i+1} be mutually incident $i-$cells of Π for all $i \in \{1, ..., n-1\}$. Then the flag $P = (C_1, C_2, ..., C_i, ..., C_n)$ is a maximal flag of Π. Since Π is regular we know that $Sym(\Pi)$ is transitive on the maximal flags of Π. Therefore if $Q = (C_1, C_2, ..., C_i', ..., C_n)$ is an adjacent flag of P then there exists a symmetry $s_i \in Sym(\Pi)$ such that $s_i P = Q$ for all $i \in \{1, ..., n\}$.
For all $j \neq i, j \in \{1, ..., n\}$ we have that C_j is left invariant by s_i. If C_j is left invariant then $s_i \in Sym(C_j)$ and then by theorem 3.3.3 also its midpoint m_j is left invariant. Here $m_1 = C_1$ since C_1 is a vertex and thus only one point.
Let m_{n+1} be the midpoint of Π, this is also left invariant since s_i is a symmetry of Π. So s_i leaves the n points $\{m_1, m_2, ..., m_{i-1}, m_{i+1}, ..., m_n, m_{n+1}\}$ unchanged.

Claim 3.3.5. These n points span a hyperplane.

We prove this claim using induction.
Base case: For a regular 2-dimensional polytope, a polygon, we have that a maximal flag consists of two elements; a vertex and an edge containing this vertex. As discussed, any symmetry that transforms a maximal flag into an adjacent flag either leaves invariant the vertex or the midpoint of the edge. In both cases the midpoint of the polygon is left invariant, so two points of interest are left invariant. Since by proposition 3.2.5 the midpoint of the polygon is not contained in any of the edges it is also not equal to any vertex. Thus we know that the two points left invariant by the symmetry span a line which is a hyperplane of the 2 dimensional space. So the points left invariant by a symmetry that transforms a maximal flag into an adjacent flag span a hyperplane.
Induction hypothesis: Suppose that for any regular $(n-1)$-dimensional polytope it holds that: if $\{m_1, m_2, ..., m_{i-1}, m_{i+1}, ..., m_{n-1}, m_n\}$ is the set of midpoints left invariant by a symmetry between two adjacent maximal flags of the polytope, then these points span a hyperplane.
Induction step: Consider a regular n-dimensional polytope G with its midpoint m_n.
Let $(D_1, ..., D_i, ..., D_n)$ be a maximal flag of G and let σ_i be the symmetry that transforms this flag into its adjacent flag $(D_1, ..., D_i', ..., D_n)$. Let m_i be the midpoint of the i-cell D_i for all $i \in \{1, ..., n\}$. Then σ_i leaves invariant all points $m_1, ..., m_{i-1}, m_{i+1}..., m_n$ as well as m_{n+1} since σ_i is a symmetry of G.
The sub-maximal flag $(D_1, ..., D_i, ..., D_{n-1})$ of G is a maximal flag of D_n. Using the induction

hypothesis on D_n we can now conclude that the points $m_1, ..., m_{i-1}, m_{i+1} ..., m_n$ span an $(n-2)$-dimensional plane, say ρ. According to proposition 3.2.5 the midpoint of G is not contained in its cell D_n and therefore not in ρ. Thus the points $m_1, ..., m_{i-1}, m_{i+1} ..., m_n, m_{n+1}$ span an affine hyperplane of $\mathbb{E}^n$.

In conclusion, the symmetry s_i leaves invariant a hyperplane thus by definition 2.4.2 we know that s_i is a reflection in this hyperplane. □

So the symmetry that transforms a maximal flag of a regular polytope into an adjacent flag is a reflection.

In the proof of the two following theorems we will see that all other symmetries are in fact a product of these reflections. For the ease of notation we will refer to the reflections reflecting a maximal flag F into an adjacent flag as the **adjacent reflections** of F.
Two symmetries $\sigma_1, \sigma_2 \in Sym(\Pi)$ of a polytope Π are called **conjugate** if there exists a third symmetry $\tau \in Sym(\Pi)$ such that $\sigma_1 = \tau \circ \sigma_2 \circ \tau^{-1}$.

Theorem 3.3.6. The adjacent reflections of any maximal flag of a regular polytope are conjugate to the adjacent reflections R_i of any initially chosen maximal flag of the polytope.

Proof. Consider a maximal flag F of the regular polytope Π. Let G also be a maximal flag of Π.
By regularity of Π we know that $Sym(\Pi)$ is transitive on the maximal flags of Π. Thus there exists a symmetry $g \in Sym(\Pi)$ such that $G = g(F)$. Let R be an adjacent reflection of F, then $F' = RF$ is an adjacent flag of F.
Consider the symmetry gRg^{-1}:

$$gRg^{-1}(G) = gRg^{-1}g(F) = gRF = g(F').$$

Then since g is a symmetry and F' is adjacent to F we have that $g(F')$ is adjacent to $g(F)$.
Thus gRg^{-1} is an adjacent reflection of G and is conjugate to R.
The adjacent reflections of a maximal flag G of a regular polytope are conjugate to the adjacent reflections of any initially chosen maximal flag F the polytope. □

It now follows that if R_i for $i = 1, ..., n$ are the adjacent reflections of some maximal flag F of a regular polytope Π then the set of symmetries $\langle R_1, \ldots, R_n \rangle$ is transitive on the maximal flags of a regular polytope.
We have that both the symmetry group of a polytope and the group generated by the reflections $R_1, \ldots, R_n$ are transitive on the maximal flags of a regular polytope. However, are they the same set? We will find that there cannot exist a symmetry in the symmetry group which is not an element of $\langle R_1, \ldots, R_n \rangle$ and thus prove the following theorem.

Theorem 3.3.7. If R_i with $i = 1, ..., n$ are the adjacent reflections of a maximal flag F of a polytope Π, then the symmetry group $Sym(\Pi)$ is generated by the reflections $R_1, ..., R_n$.

Proof. Let Π be a regular polytope and let F be a maximal flag of Π. We know there exists some $\sigma \neq I, \sigma \in Sym(\Pi)$ such that it will map a maximal flag G onto F, $\sigma(G) = F$. By property 2.3.5 we know that there exists some sequence of flags $(F = F_1, F_2, ..., F_k = G)$ such that F_i and F_{i+1} are adjacent flags for all $i \in \{1, ..., k-1\}$. For every pair of F_i and F_{i+1} theorem 3.3.4 says that there exists a reflection, say H_i, such that $H_i F_i = F_{i+1}$.

So $H_1F = F_2$ and $H_2F_2 = F_3$, those two equalities can be combined to find that $H_2H_1F = F_3$. Successively applying this to the sequence $(F = F_1, F_2, ..., F_k = G)$ we find

$$H_{k-1}H_{k-2}...H_1F = G. \tag{3.1}$$

Moreover, using $\sigma(G) = F$ we find

$$H_{k-1}H_{k-2}...H_1\sigma(G) = G. \tag{3.2}$$

From this we know that $H_{k-1}H_{k-2}...H_1\sigma$ leaves G invariant and is therefore the identity. Thus the symmetry σ is the inverse of the product of reflections $H_{k-1}H_{k-2}...H_1$, $\sigma = H_1^{-1}...H_{k-2}^{-1}H_{k-1}^{-1}$. A reflection is its own inverse, hence σ is a product of the reflections $H_1, ..., H_k$.
We will now prove by induction that the reflections H_i are elements of the group generated by the reflections $R_1, ..., R_n$.
Base case: F_2 is adjacent to $F_1 = F$. Suppose $H_1 = R_1$ then $F_2 = R_1F_1$, then by theorem 3.3.6 the adjacent reflections of F_2 are $R_1R_iR_1$ for $i = 1, ..., n$. So $H_2 = R_1R_iR_1$ for some $i \in \{1, ..., n\}$ and thus $H_2 \in \langle R_1, ..., R_n\rangle$.
Induction hypothesis: Suppose that for some $j \in \{2, ..., k-2\}$ the adjacent reflections of F_j are given by $S_jR_iT_j$ for $i = 1, ..., n$ with $S_j = H_{j-1}...H_1$ and $T_j = H_1...H_{j-1}$ and $H_i \in \langle R_1, ..., R_n\rangle$ for $i = 1, ..., j$.
Induction step: Consider the maximal flag F_{j+1}. We have $F_{j+1} = H_jF_j$, then by theorem 3.3.6 the adjacent reflections of F_{j+1} are $H_jS_jR_iT_jH_j$ for $i = 1, ..., n$. Thus we have $H_{j+1} = H_jH_{j-1}...H_1R_iH_1...H_{j-1}H_j$ for some unique $i \in \{1, ..., n\}$. Therefore $H_{j+1} \in \langle R_1, ..., R_n\rangle$.

In consequence of σ being the product $H_1H_2...H_k$ we also have that σ is in the group generated by the reflections $R_1, ..., R_n$, $\sigma \in \langle R_1, \ldots, R_n\rangle$. Hence the entire symmetry group $Sym(\Pi)$ is generated by the reflections $R_1, ..., R_n$. $\square$

To conclude this paragraph, we have seen that every regular polytope admits n adjacent reflections for any of its maximal flags. If one of these flags is chosen as a base flag with its adjacent reflections R_i for $i = 1, \ldots, n$ then the adjacent reflections of any other maximal flag of the polytope are generated by the reflections $R_1, ..., R_n$. Thus the group $\langle R_1, \ldots, R_n\rangle$ is transitive on the maximal flags of that regular polytope. From this it follows that the symmetry group of the polytope is generated by $R_1, ..., R_n$.

Chapter 4

Properties of symmetries

The symmetry group of a regular n-dimensional polytope is generated by the adjacent reflections R_i for $i = 1, ..., n$ of some maximal flag of the polytope. There are some nice properties based on these reflections which we will be presenting in this chapter. We will discuss the existence of fundamental regions and discuss a notation for a regular polytope by a diagram or its Schläfli symbol. We will prove that up to isomorphisms there exists only a limited number of regular polytopes for any dimension greater than two. Furthermore, it will be shown that it is possible to construct a regular polytope given such a diagram.

4.1 Fundamental Region

Let G be a group of isometries of $\mathbb{E}^n$ fixing the origin O. A **fundamental region** of G is a subset of a sphere centered in O such that the images of the subset under the symmetry group just cover the sphere without overlapping except for the points on the boundaries. This means every point on the sphere is equivalent (under the symmetry group) to some point of the fundamental region, but no two points of the region are equivalent unless both are on the boundary [2]. Here two points a and b are said to be **equivalent** if there exists a symmetry σ in G such that $\sigma(a) = b$. As a consequence we have that every point of the sphere belongs to one unique fundamental region of G, except for the boundary points of a fundamental region.
Furthermore, a fundamental region is only left invariant by the identity.

Theorem 4.1.1. For every finite group of isometries of $\mathbb{E}^n$ fixing O there exist fundamental regions.

Proof. Let G be the symmetry group generated by some finite group of isometries of $\mathbb{E}^n$ fixing O. Let $B \in \mathbb{E}^n$ be a sphere centered in O with some radius r, $B = \{\vec{x} \in \mathbb{E}^n : |\vec{x} - O| = r\}$. The **orbit** T of some point $v \in B$ is the set of images $g(v)$ of v for all symmetries $g \in G$, it holds $g(v) \in B$. If for $v_1, v_2 \in B$ there does not exist a symmetry $f \in G$ such that $f(v_1) = v_2$ (or vice versa), then the orbits T_1 and T_2 of v_1 and v_2 respectively do not have any points in common. Suppose T_1, T_2 have a point u in common. Then there exist symmetries h and k such that $h(v_1) = u = k(v_2)$, but then we also have $k^{-1}(h(v_1)) = k^{-1}(u) = v_2$, where k^{-1} is the inverse of k. Which contradicts there not being a symmetry f such that $f(v_1) = v_2$. Thus we know that every point of B is in exactly one orbit. If we consider the orbit for all points of B then we have all possible orbits. These orbits exactly cover the full sphere.
Let the set of points S contain exactly one point from every orbit. Then every image of S under symmetry will contain exactly one point from every orbit as well. All these images will cover the orbits and therefore cover the sphere without overlap. In other words S is a fundamental region of the symmetry group G. □

The images of a fundamental region cover the sphere and are by definition congruent to the fundamental region. What does this tell us about the images?

Lemma 4.1.2. The image of a fundamental region under symmetry is also a fundamental region of the symmetry group.

Proof. Let G be the symmetry group generated by some finite group of isometries of $\mathbb{E}^n$ fixing O. Let $B \in \mathbb{E}^n$ be a sphere centered in O with some radius r, $B = \{\vec{x} \in \mathbb{E}^n : |\vec{x} - O| = r\}$.
Let F be a fundamental region of G. If p is a point in B, then there exists some symmetry $h \in G$ such that $p \in h(F)$. Consider the image $g(F)$ of F under the symmetry $g \in G$. Let g^{-1} be the inverse of g, then $p \in h(g^{-1}(g(F)))$ where $h(g^{-1}(g(F)))$ is also an image of $g(F)$. So p is a point in the image of $g(F)$ under the symmetry $h(g^{-1})$.
In conclusion, for any pair $(g(F), h(F))$ of images of F it holds: There exists a symmetry $\sigma \in G$, such that $\sigma(g(F)) = h(F)$. Thus the symmetry group G acts transitively on the orbit of a fundamental region.
Note that it now holds that for all symmetries $g \in G$, the images of $g(F)$ also just cover the sphere B without overlapping since they are the same images as the ones of F. Therefore for all symmetries $g \in G$ the image $g(F)$ of F is also a fundamental region of the symmetry group. □

Lemma 4.1.3. Let F be a fundamental region of some symmetry group G and let $\mathcal{F} = \{\tau(F) : \tau \in G\}$ be the set of images of F under the symmetry group G. If F_1 and F_2 are both elements of $\mathcal{F}$, then there exists a unique symmetry $\sigma \in G$ such that $\sigma(F_1) = F_2$.

Proof. Let G be the symmetry group generated by some finite group of isometries of $\mathbb{E}^n$ fixing O. Let $B \in \mathbb{E}^n$ be a sphere centered in O with some radius r, $B = \{\vec{x} \in \mathbb{E}^n : |\vec{x} - O| = r\}$.
Let $\mathcal{F}$ be as stated. Let F_1 and F_2 be elements of $\mathcal{F}$. There exist $\sigma_1, \sigma_2 \in g$ such that $F_i = \sigma_i(F)$ for $i = 1, 2$. By lemma 4.1.2 F_1 and F_2 are also fundamental regions and

$$\sigma_2(\sigma_1^{-1}(F_1)) = \sigma_2(\sigma_1^{-1}(\sigma_1(F))) = \sigma_2(F) = F_2. \tag{4.1}$$

Thus there exists a symmetry $\sigma \in G$ such that $\sigma(F_1) = F_2$, namely $\sigma = \sigma_2\sigma_1^{-1}$.
Suppose p is a point on the sphere B, by definition of the fundamental region F it holds that there exists a $\tau \in G$ with $p \in \tau(F) \in \mathcal{F}$ and $B = \bigcup\{\tau(F) : \tau \in Sym(\Pi)\}$.
Suppose $p \in F_1 \cap F_2$. Then $p \in \sigma_1(F)$, $p \in \sigma_2(F)$. We have

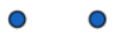
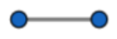
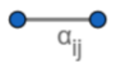
$$\sigma_1\sigma_2^{-1}(p) \in \sigma_1\sigma_2^{-1}(F_2) = \sigma_1(F) = F_1. \tag{4.2}$$

To conclude, $\sigma_1\sigma_2^{-1}(p) \in F_1$ and $p \in F_1$. But $F_1 \cap \tau(F_1)$ is either an empty set ($\emptyset$) or it is simply F_1. Thus we find that $\sigma_1\sigma_2^{-1}$ is the identity symmetry I, since by definition a fundamental region is only left invariant under the identity. Therefore σ_2^{-1} is the inverse of σ_1 and $\sigma_1 = \sigma_2$. Which proves that p can only be a point in exactly one image of the fundamental region F. Thus the symmetry $\sigma \in G$ such that $\sigma(F_1) = F_2$ is unique. □

We have established the existence of fundamental regions for symmetry groups fixing some point O with some of its features.

4.2 Representation by diagrams

In chapter 3.3 we have seen that there is a set of reflections which generates the symmetry group of a regular polytope Π. Let $F = (C_1, .., C_n)$ be a maximal flag of Π, then the adjacent reflections of F generate the symmetry group of Π. In this chapter we will denote these reflections by $R_1, \ldots, R_n$ and the corresponding reflection axes by $r_1, \ldots, r_n$.
Let $R_j r_i$ be the hyperplanes representing the images after reflecting r_i by R_j for $i, j \in \{1, ..., n\}$. Consider the set of hyperplanes $S = \{\sigma(r_i) : \sigma \in \langle R_1, \ldots, R_n \rangle, i = 1, ..., n\}$. Every hyperplane in this set is an axis of reflection of the polytope. The midpoint of the polytope is left invariant by all reflections $R_1, \ldots, R_n$ and is therefore contained in each of the hyperplanes in S.
A **unit sphere** is a sphere with a radius of size 1. In the n-dimensional space we consider a unit sphere U to be the set of vectors $\vec{x} \in \mathbb{E}^n$ all emanating from a center vector $\vec{c} \in \mathbb{E}^n$ satisfying $|\vec{x} - \vec{c}| = 1$, $U = \{\vec{x}, \vec{c} \in \mathbb{E}^n : |\vec{x} - \vec{c}| = 1\}$. If we consider the unit sphere centered in the common point of the hyperplanes, then these planes cut out a pattern of great circles decomposing the spherical surface into a number of congruent regions, let Z be the smallest of these regions. For the internal angles of Z it will be sufficient to consider real divisors of π; for if the angle is $\frac{j\pi}{p}$, where j and p are co-prime, we can find a multiple of j/p which differs from $1/p$ by an integer and hence it is equivalent to some given angle π/p [2].This means we can find a multiple of j/p such that the angle $\frac{j\pi}{p}$ yields the same result as the angle π/p.
Thus the possible angle values are $\frac{\pi}{2}, \frac{\pi}{3}, \ldots$. For each hyperplane separating two congruent regions on the sphere it holds that the two regions are each others image upon reflecting in this hyperplane. Because of this, every point on the sphere is equivalent (by the action of the group) to some point belonging to the region Z. The spherical surface is decomposed into regions congruent to Z, so the images of Z cover the sphere while only overlapping on their boundaries. Therefore Z is a fundamental region for the symmetry group.
The various possible fundamental regions are very conveniently classified by associating them with certain graphs, called **diagrams**. The nodes in such diagrams represent the hyperplanes corresponding to the generating reflections. Two nodes are joined by an edge whenever the corresponding hyperplanes are not perpendicular to each other. We mark the edge connecting nodes i and j for all $i, j \in \{1, ..., n\}$ with a number $\alpha_{ij} \geq 3$ to indicate that the product of reflections $R_i R_j$ has order α_{ij}. Since the order 3 so frequently occurs it is usually omitted. Following this structure, here is how we denote two hyperplanes with their α values:

$\alpha_{ij} = 1$	$\alpha_{ij} = 2$	$\alpha_{ij} = 3$	$\alpha_{ij} > 3$
			

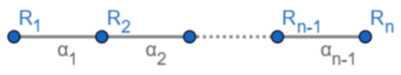
When $\alpha_{ij} = 1$ we have one node for two reflection hyperplanes since the hyperplanes are identical if the product of their reflections has order 1. Moreover, two unconnected nodes denote commutative reflections or perpendicular mirrors. Two nodes joined by an unmarked edge denote the reflections R_i and R_j that satisfy

$$R_i R_j R_i = R_j R_i R_j \tag{4.3}$$

and two nodes joined by a edge marked α denote the reflections that satisfy

$$(R_i R_j)^{\alpha} = I. \tag{4.4}$$

Theorem 4.2.1. If Π is an n-dimensional regular polytope, then there are n reflections $R_1, R_2, ..., R_n$ with $Sym(\Pi) = \langle R_1, R_2, ..., R_n \rangle$ and $R_1, R_2, ..., R_n$ have diagram:



With α_i being the order of the product of the reflections R_i and R_{i+1}.

Proof. Let Π be a regular polytope contained in $\mathbb{E}^n$ and let $F = (C_1, .., C_n)$ be a maximal flag of Π. If $R_1, \ldots, R_n$ are the reflections that reflect F into its adjacent flags such that $R_i F = F_i$, then by theorem 3.3.7 it holds that $Sym(\Pi) = \langle R_1, ..., R_n \rangle$.
Let us consider the two reflections R_j and R_k for $j \neq k$, such that $R_j F = (C_1, \ldots, C'_j, \ldots, C_n)$ and $R_k F = (C_1, \ldots, C'_k, \ldots, C_n)$. We distinguish the two cases: $\begin{cases} |j-k|> 1 \\ |j-k|= 1 \end{cases}$
The numbers j and k are arbitrarily chosen so without loss of generality we may assume that $k > j$.

If $|j-k|> 1$, then we know that R_k leaves every element of F invariant except for the k-cell C_k. More precisely, R_k leaves C_{j-1}, C_j and C_{j+1} invariant and then because of the diamond property 2.3.4 also C'_j is left invariant by the reflection R_k. Now we find:

$$R_k F_j = R_k R_j F = (C_1, \ldots, C'_j, \ldots, C'_k, \ldots, C_n). \tag{4.5}$$

Following the same reasoning we find that R_j leaves C'_k invariant since it leaves C_{k-1}, C_k and C_{k+1} invariant, thus:

$$R_j F_k = R_j R_k F = (C_1, \ldots, C'_j, \ldots, C'_k, \ldots, C_n). \tag{4.6}$$

From equations 4.5 and 4.6 we conclude that $R_k R_j F = R_j R_k F \Rightarrow R_k R_j = R_j R_k$, therefore

$$(R_k R_j)^2 = I. \tag{4.7}$$

The hyperplanes corresponding to the reflections R_j and R_k are perpendicular to each other and therefore the pair of nodes in the diagram representing R_j and R_k are not connected by an edge if $|j-k|> 1$.

What if $|j-k|= 1$, then $R_k = R_{j+1}$ and we have the flags:

- $F = (C_1, \ldots, C_j, C_{j+1}, \ldots, C_n)$
- $F_j = (C_1, \ldots, C'_j, C_{j+1}, \ldots, C_n)$
- $F_{j+1} = (C_1, \ldots, C_j, C'_{j+1}, \ldots, C_n)$

So we know $C_j \subseteq C_{j+1}$, $C'_j \subseteq C_{j+1}$, $C_j \subseteq C'_{j+1}$. Now suppose that the hyperplanes corresponding to the reflections R_j and R_{j+1} are perpendicular. Then $(R_j R_{j+1})^2 = I$ and

$$R_{j+1} R_j F = (C_1, \ldots, C'_j, C'_{j+1}, \ldots, C_n) = R_j R_{j+1} F. \tag{4.8}$$

This implies that the flags containing $(C_1, ..., C_{j-1}, C_{j+2}, ..., C_n)$ form a 2-gon. However by the definition of a polytope they should form a polygon which by definition is not defined on two vertices. Thus we have a contradiction.
In the case of $|j-k|= 1$ the nodes in the diagram representing the reflections R_j and R_k will be connected by an edge. $\square$

For the ease of notation we will denote such diagrams by its consecutive α_i's: $\{\alpha_1, \alpha_2, ..., \alpha_{n-1}\}$. This notation is called a **Schläfli symbol**. If the Schläfli symbol $\{\alpha_1, \alpha_2, ..., \alpha_{n-1}\}$ belongs to the generating reflections of a polytope Π then the polytope with generating reflections belonging to the Schläfli symbol $\{\alpha_{n-1}, ..., \alpha_2, \alpha_1\}$ is called the **dual** of Π. Moreover, if $\{\alpha_1, \alpha_2, ..., \alpha_{n-1}\} = \{\alpha_{n-1}, ..., \alpha_2, \alpha_1\}$ then Π is called **self-dual**.

4.2.1 Possible diagrams

So we have established that for every regular polytope there exists a Schläfli symbol that represents the diagram belonging to the generating reflections of that polytope. We will use this knowledge to establish all possible values for α_{ij}. Doing so we will find that there only exist a finite number of these polytopes in three dimensions and higher.

For a regular polygon the general Schläfli symbol, belonging to the diagram of the generating reflections of this polygon, is $\{p\}$. Here $p > 2$, for if $p = 1$ we find the hyperplanes to be equal and if $p = 2$ then the reflections would commute, this means we will have reflections in two planes perpendicular to each other. From $\{p\}$ we can deduce that there are two generating reflections R_1, R_2 from which the reflection axes are inclined at an angle of $\frac{q\pi}{p}$ with $\gcd(p, q) = 1$.
The product R_1R_2 of the reflection in the axes of R_2 and R_1, in that order, is the rotation over the angle $\frac{2q\pi}{p}$. Rotating p times over this angle is equal to the identity, since after p rotations we made a rotation over $2q\pi$ radians which is q times a full turn. So the product R_1R_2 is of order p.
We know that one of the two reflections leaves invariant the vertex of a maximal flag. Rotating this vertex p times over the angle π/p gives us all images of the vertex. So the polygon generated by the reflections R_1, R_2 is a polygon with p vertices, it is a p-gon.
For a regular polytope in 2-dimensional space we have found that the possible diagrams are denoted by the Schläfli symbols $\{3\}, \{4\}, ..., \{p\}$.

If for the angle $\frac{q\pi}{p}$ with $\gcd(p, q) = 1$ between the reflection axes of any pair of generating reflections we have $q \neq 1$ then we say that the polytope constructed from these generating reflections is a **stellated** polytope.
A stellated polytope is closely related to some polytope, it exists of the same vertices. However the edges, faces and cells connect different sets of vertices and therefore the representational polytope looks very different.
An example of a stellation of a polytope is that of the pentagon. Both the pentagon and its stellation correspond to the Schläfli symbol $\{5\}$. However, the reflection axes of the generating reflections for the pentagon are inclined at an angle of $\frac{\pi}{5}$ while for the stellation they are inclined at an angle of $\frac{2\pi}{5}$.
The stellation of a pentagon is depicted in figure 4.1.

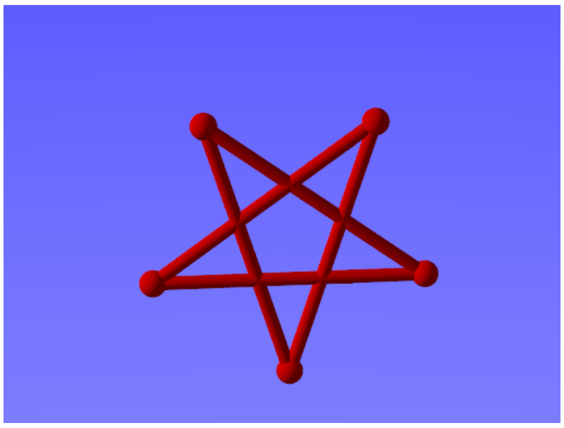


Figure 4.1: A stellated pentagon

Furthermore, we will take a look into the possible Schläfli symbols for a regular polyhedron.

Theorem 4.2.2. There exist 5 Schläfli symbols for regular polyhedra (up to symmetries): $\{3,3\},\{3,4\},\{4,3\},\{3,5\},\{5,3\}$.

Proof. Consider the symmetry group G generated by three reflections R_i for $i = 1,2,3$ in $\mathbb{E}^3$ corresponding to a Schläfli symbol $\{p,q\}$. Let O be the intersection of the three reflection axes V_i that correspond to the reflections R_i for $i \in \{1,2,3\}$ and let $B \in \mathbb{E}^3$ be a sphere centered in O with some radius r, $B = \{\vec{x} \in \mathbb{E}^n : |\vec{x} - O| = r\}$.
The intersections of the hyperplanes in the set $S = \{\sigma(r_i) : \sigma \in \langle R_1, R_2, R_3\rangle\}$ with B are circular arcs.
Consider the smallest triangle on the sphere formed by these arcs. The angles of this triangle are $\frac{\pi}{p}, \frac{\pi}{q}, \frac{\pi}{2}$. On the sphere we have

$$\frac{\pi}{p} + \frac{\pi}{q} + \frac{\pi}{2} > \pi.$$

Where $p,q > 2$ and $p,q \in \mathbb{N}$. Under these conditions there are only 5 cases for which the inequality is satisfied. Thus we find the five Schläfli symbols for the regular polyhedra [2]:
$\{3,3\},\{3,4\},\{4,3\},\{3,5\},\{5,3\}$. □

For the Schläfli symbol of a general n-dimensional regular polytope we state the following theorem:

Theorem 4.2.3. The α_i's in a diagram for the reflections $R_1, ..., R_n$ of a regular n-dimensional polytope have limited values for $n > 2$. The Schläfli symbols for each dimension are:

n	value combinations
2	$\{3\},\{4\},...,\{p\}$
3	$\{3,3\},\{3,4\},\{4,3\},\{3,5\},\{5,3\}$
4	$\{3,3,3\},\{3,3,4\},\{4,3,3\},\{3,4,3\},\{3,3,5\},\{5,3,3\}$
5	$\{3,3,3,3\},\{4,3,3,3\},\{3,3,3,4\}$
...	
k	$\{3,3,...,3\},\{4,3,...,3\},\{3,3,...,3,4\}$

Proof. Consider the symmetry group $\langle R_1, ..., R_n\rangle$ generated by the reflections $R_1, ..., R_n$ in $\mathbb{E}^n$ that satisfy a diagram corresponding to the Schläfli symbol $\{\alpha_1, ..., \alpha_{n-1}\}$. Let $O \in \mathbb{E}^n$ be the intersection point of the reflection axes of the reflections $R_1, ..., R_n$, so each reflection R_i leaves O invariant.
Let $B = \{\vec{x} \in \mathbb{E}^n : |\vec{x} - O|= 1\}$ be the unit sphere centered in O. Let $r_1, ..., r_n$ be the affine hyperplanes that enclose the smallest fundamental region for the symmetry group $\langle R_1, ..., R_n\rangle$ on B. Let a_i be a normal vector of the hyperplane r_i for every $i \in \{1, ..., n\}$. Let Π be a regular n-dimensional polytope with $Sym(\Pi) = \langle R_1, ..., R_n\rangle$.

Lemma 4.2.4. The vectors $\vec{a}_i$ are linearly independent.

Proof. By definition we know that $Sym(\Pi)$ is transitive on the elements of Π. Let v be a vertex of Π, then applying all possible products of the reflections R_i yields us all vertices of the polytope because of transitivity. A reflection image of v is given by the formula

$$R(\vec{v}) = \vec{v} - 2\langle\vec{v}|\vec{n}\rangle\vec{n}.$$

Thus by reflecting our vertex v we only translate v by some linear combinations of the normal vectors $\vec{a}_i$. These vectors $\vec{a}_i$ span a space of dimension n since we have all the vertices of an n-dimensional polytope.
Now suppose the vectors $\vec{a}_i$ are not linearly independent, then at most $n-1$ of the $\vec{a}_i$'s are independent and thus the set of vectors $\vec{a}_1, ..., \vec{a}_n$ span a subspace W such that the dimension of W is at most $n - 1$, $dim(W) \leq n - 1$. Which is a contradiction.
Thus the normal vectors $\vec{a}_1, ..., \vec{a}_n$ are linearly independent. □

Let the lengths of the normal vectors $\vec{a}_i$ be $\sqrt{2}$, $|\vec{a}_i|= \sqrt{2}$.

$$\langle\vec{a}_i|\vec{a}_i\rangle = |\vec{a}_i|^2= 2 \tag{4.9}$$

For every pair of vectors $\vec{a}_i, \vec{a}_j$ such that $j > i + 1$ or $i > j + 1$ then the corresponding hyperplanes r_i, r_j are perpendicular so

$$\langle\vec{a}_i|\vec{a}_j\rangle = 0. \tag{4.10}$$

If $j = i + 1$ or $i = j + 1$ then the corresponding hyperplanes r_i, r_j are induced at an angle $\frac{\pi}{\alpha_i}$ and so

$$\langle\vec{a}_i|\vec{a}_j\rangle = \pm 2 \cdot \cos\left(\frac{\pi}{\alpha_i}\right). \tag{4.11}$$

If $+\vec{a}_i^*$ is a normal vector of r_i then so is $-\vec{a}_i^*$. Let $\vec{a}_i = \pm\vec{a}_i^*$, then the $\pm$ sign in equation 4.11 is depending on our choices for the vectors $\vec{a}_i$.
Suppose for some $j \in \{1, ..., n - 1\}$ we have $\vec{a}_{j+1} = +\vec{a}_{j+1}^*$ and $\langle\vec{a}_j|\vec{a}_{j+1}\rangle = -2 \cdot \cos\left(\frac{\pi}{\alpha_i}\right)$, if we were to use $\vec{a}_{j+1} = -\vec{a}_{j+1}^*$ instead of $\vec{a}_{j+1} = +\vec{a}_{j+1}^*$ then $\langle\vec{a}_j|\vec{a}_{j+1}\rangle = 2 \cdot \cos\left(\frac{\pi}{\alpha_i}\right)$.
Consider $\vec{a}_1$, let us choose $\vec{a}_2 = \pm\vec{a}_2^*$ such that $\langle\vec{a}_1|\vec{a}_2\rangle = 2 \cdot \cos\left(\frac{\pi}{\alpha_i}\right)$ then $\vec{a}_1$ and $\vec{a}_2$ are fixed. Now we can choose $\vec{a}_3 = \pm\vec{a}_3^*$ such that $\langle\vec{a}_2|\vec{a}_3\rangle = 2 \cdot \cos\left(\frac{\pi}{\alpha_i}\right)$, and so on. Eventually we have chosen $\vec{a}_n$ and now all $\vec{a}_i$ are fixed such that

$$\langle\vec{a}_i|\vec{a}_{i+1}\rangle = 2 \cdot \cos\left(\frac{\pi}{\alpha_i}\right) \text{ for all } i = 1, ..., n - 1. \tag{4.12}$$

Definition 4.2.5. A **Gram matrix** of a set of vectors is the matrix of inner products whose entries are given by $G_{ij} = \langle \vec{a}_i | \vec{a}_j \rangle$ for all $i, j \in \{1, ..., n\}$.
The gram matrix G of an $m \times n$-matrix A is given by $A^{\intercal}A$ [10].

- G is always a square matrix.
- G is always symmetric and positive semi-definite.
- The rank of G is the same as the rank of A.

With the above information and definition we can construct the Gram matrix G corresponding to the vectors $\vec{a}_i$ for $i = 1, ..., n$.

$$G = \begin{pmatrix} 2 & 2 \cdot \cos\left(\frac{\pi}{\alpha_1}\right) & 0 & \dots & 0 \\ 2 \cdot \cos\left(\frac{\pi}{\alpha_1}\right) & 2 & \ddots & \ddots & \vdots \\ 0 & \ddots & \ddots & \ddots & 0 \\ \vdots & \ddots & \ddots & 2 & 2 \cdot \cos\left(\frac{\pi}{\alpha_{n-1}}\right) \\ 0 & \dots & 0 & 2 \cdot \cos\left(\frac{\pi}{\alpha_{n-1}}\right) & 2 \end{pmatrix} \tag{4.13}$$

The matrix G is **positive definite** if $\vec{x}^{\intercal}G\vec{x} > 0$ for all $\vec{x} \in \mathbb{E}^n \setminus \{0\}$. The vectors $\vec{a}_i$ with $i = 1, ..., n$ are linearly independent by lemma 4.2.4 and thus form a basis for $\mathbb{E}^n$. Now for any nonzero vector $\vec{v} = \sum_{j=1}^{n} v_j \vec{a}_j$, we have

$$\vec{v}^{\intercal}G\vec{v} = \sum_{j=1}^{n}(\sum_{i=1}^{n} v_i \cdot \langle \vec{a}_i | \vec{a}_j \rangle) \cdot v_j = \sum_{j=1}^{n} \langle \sum_{i=1}^{n} v_i \cdot \vec{a}_i | \vec{a}_j \rangle \cdot v_j = \sum_{j=1}^{n} \langle \vec{v} | \vec{a}_j \rangle \cdot v_j = \langle \vec{v} | \sum_{j=1}^{n} \vec{a}_j \cdot v_j \rangle = \langle \vec{v} | \vec{v} \rangle > 0.$$

Thus G is positive definite. For a positive definite matrix we know from [6] that given a set $J \subseteq \{1, ..., n\}$, if $G_J := (g_{ij})_{i,j \in J}$ denotes the submatrix of G restricted to the rows and columns indexed by J then $det(G_J) > 0$. Where $det(G_J)$ is the determinant of the matrix G_J.
Such a submatrix G_J for $J := \{k, ..., l\}$ is given by

$$G_J = \begin{pmatrix} 2 & 2 \cdot \cos\left(\frac{\pi}{\alpha_k}\right) & 0 & \dots & 0 \\ 2 \cdot \cos\left(\frac{\pi}{\alpha_k}\right) & 2 & \ddots & \ddots & \vdots \\ 0 & \ddots & \ddots & \ddots & 0 \\ \vdots & \ddots & \ddots & 2 & 2 \cdot \cos\left(\frac{\pi}{\alpha_{l-1}}\right) \\ 0 & \dots & 0 & 2 \cdot \cos\left(\frac{\pi}{\alpha_{l-1}}\right) & 2 \end{pmatrix} \tag{4.14}$$

and thus has the same structure as the Gram matrix for a Schläfli symbol of length $l - k$. Thus the $m - 1$ alpha values of any connected sub-diagram of length m of the diagram of the reflections $R_1, ..., R_n$ must also occur among the values for the Schläfli symbols for a diagram of length m.

Moreover, we find a relation between the determinants of a Gram matrix and some of its submatrices. Namely, let G_k be the Gram matrix of a diagram with k nodes and the edges labeled by $\alpha_1, ..., \alpha_{k-1}$ then we find

$$det(G_k) = 2 \cdot det(G_{k-1}) - 4 \cdot \cos^2(\pi/\alpha_1) \cdot det(G_{k-2}). \tag{4.15}$$

In this relation we consider G_{k-1} and G_{k-2} to be the Gram matrix on the $k-1$ and $k-2$ nodes respectively with the edges labeled $\alpha_2, ..., \alpha_{k-1}$ and $\alpha_3, ..., \alpha_{k-1}$.

For $n = 2, 3$ we have already seen the possible value combinations in the diagram of a regular polytope. Consider $n = 4$, we then have a Schläfli symbol $\{p, q, r\}$ where both $\{p, q\}$ and $\{q, r\}$ must occur among the Schläfli symbols for $n = 3$.
By this reasoning we find the following possible Schläfli symbols:

$$\{3,3,3\}, \{3,3,4\}, \{4,3,3\}, \{3,4,3\}, \{3,3,5\}, \{5,3,3\},$$
$$\{4,3,4\}, \{3,5,3\}, \{5,3,4\}, \{4,3,5\}, \{5,3,5\}. \tag{4.16}$$

So the possibilities are already limited. Since the hyperplanes in each of the pairs $(r_1, r_3), (r_1, r_4), (r_2, r_4)$ are perpendicular to each other we have that $\langle \vec{a}_1 | \vec{a}_3 \rangle = \langle \vec{a}_1 | \vec{a}_4 \rangle = \langle \vec{a}_2 | \vec{a}_4 \rangle = 0$. The vectors $\vec{a}_i$ with $i = 1, ..., 4$ have $|\vec{a}_i| = \sqrt{2}$ so from the Schläfli symbol we get the following relations:

$$\langle \vec{a}_i | \vec{a}_i \rangle = |\vec{a}_i|^2 = 2$$

$$\langle \vec{a}_1 | \vec{a}_2 \rangle = 2 \cdot \cos\left(\frac{\pi}{p}\right)$$

$$\langle \vec{a}_2 | \vec{a}_3 \rangle = 2 \cdot \cos\left(\frac{\pi}{q}\right)$$

$$\langle \vec{a}_3 | \vec{a}_4 \rangle = 2 \cdot \cos\left(\frac{\pi}{r}\right)$$

With this information we can construct the Gram matrix G.

$$G = \begin{pmatrix} 2 & 2 \cdot \cos\left(\frac{\pi}{p}\right) & 0 & 0 \\ 2 \cdot \cos\left(\frac{\pi}{p}\right) & 2 & 2 \cdot \cos\left(\frac{\pi}{q}\right) & 0 \\ 0 & 2 \cdot \cos\left(\frac{\pi}{q}\right) & 2 & 2 \cdot \cos\left(\frac{\pi}{r}\right) \\ 0 & 0 & 2 \cdot \cos\left(\frac{\pi}{r}\right) & 2 \end{pmatrix}$$

According to relation 4.15 the determinant of G is

$$det(G) = 2 \cdot (8 - 8\cos^2(\frac{\pi}{q}) - 8\cos^2(\frac{\pi}{r})) - 4\cos^2(\frac{\pi}{p})(4 - 4\cos^2(\frac{\pi}{r})).$$

We have the possibilities given in equation 4.16 and for the determinant of the Gram matrix we have $det(G) > 0$ since the submatrix $G_J = G$ if $J = \{1, ..., 4\}$. So we check if the possibilities from 4.16 satisfy the inequality $det(G) > 0$.

Schläfli symbol	***det(G)***
$\{3,3,3\}$	5
$\{3,3,4\}$	2
$\{4,3,3\}$	2
$\{3,4,3\}$	1
$\{3,3,5\}$	$\frac{1}{2}(7-3\sqrt{5}) \approx 0.145$
$\{5,3,3\}$	$\frac{1}{2}(7-3\sqrt{5}) \approx 0.145$
$\{4,3,4\}$	0
$\{3,5,3\}$	$3-2\sqrt{5} \approx -1.472$
$\{5,3,4\}$	$1-\sqrt{5} \approx -1.236$
$\{4,3,5\}$	$1-\sqrt{5} \approx -1.236$
$\{5,3,5\}$	$\frac{7}{2}-\frac{5\sqrt{5}}{2} \approx -2.090$

For the values of the last five Schläfli symbols we see that the determinant of G is not greater than zero. So for these possibilities G is not positive definite as it should be and thus these possibilities do not satisfy the requirements. As a result we find that the Schläfli symbols for a 4-dimensional regular polytope are

$$\{3,3,3\}, \{3,3,4\}, \{4,3,3\}, \{3,4,3\}, \{3,3,5\}, \{5,3,3\}.$$

Here $\{3,3,4\}$ and $\{4,3,3\}$ are closely related to each other since if we have the hyperplanes $r_1, ..., r_4$ satisfying one of these symbols then the order of the hyperplanes can be reversed to satisfy the reversed symbol. This also holds for $\{3,3,5\}$ and $\{5,3,3\}$. In chapter 4.2.2 it will become clear that the ordering of the hyperplanes is of importance and thus we already mention both possibilities.
We will take a look at the case $n = 5$. If we consider the Schläfli symbol $\{p,q,r,s\}$ then $\{p,q,r\}$ and $\{q,r,s\}$ need to occur among the Schläfli symbols for $n = 4$, we find the following possibilities for $n = 5$

$$\{3,3,3,3\}, \{3,3,3,4\}, \{4,3,3,3\}, \{3,3,4,3\}, \{3,4,3,3\}, \{4,3,3,4\},$$
$$\{3,3,3,5\}, \{5,3,3,3\}, \{5,3,3,4\}, \{4,3,3,5\}, \{5,3,3,5\}. \tag{4.17}$$

Furthermore let H be the Gram matrix for $n = 5$, we have

$$H = \begin{pmatrix} 2 & 2\cdot\cos\left(\frac{\pi}{p}\right) & 0 & 0 & 0 \\ 2\cdot\cos\left(\frac{\pi}{p}\right) & 2 & 2\cdot\cos\left(\frac{\pi}{q}\right) & 0 & 0 \\ 0 & 2\cdot\cos\left(\frac{\pi}{q}\right) & 2 & 2\cdot\cos\left(\frac{\pi}{r}\right) & 0 \\ 0 & 0 & 2\cdot\cos\left(\frac{\pi}{r}\right) & 2 & 2\cdot\cos\left(\frac{\pi}{s}\right) \\ 0 & 0 & 0 & 2\cdot\cos\left(\frac{\pi}{s}\right) & 2 \end{pmatrix}$$

Using relation 4.15 we have that

$$det(H) = 2\cdot(2\cdot(8-8\cos^2(\frac{\pi}{r})-8\cos^2(\frac{\pi}{s}))-4\cos^2(\frac{\pi}{q})(4-4\cos^2(\frac{\pi}{s})))-4\cdot\cos^2(pi/p)\cdot(8-8\cos^2(\frac{\pi}{r})-8\cos^2(\frac{\pi}{s}))$$

Again we want $det(H) > 0$ and thus check the values of $det(H)$ for each possibility:

Schläfli symbol	***det(G)***
$\{3,3,3,3\}$	6
$\{3,3,3,4\}$	2
$\{4,3,3,3\}$	2
$\{3,4,3,3\}$	0
$\{3,3,4,3\}$	0
$\{4,3,3,4\}$	0
$\{3,3,3,5\}$	$4-2\sqrt{5}\approx -0.472$
$\{5,3,3,3\}$	$4-2\sqrt{5}\approx -0.472$
$\{5,3,3,4\}$	$1-\sqrt{5}\approx -1.236$
$\{4,3,3,5\}$	$1-\sqrt{5}\approx -1.236$
$\{5,3,3,5\}$	$5-3\sqrt{5}\approx -2.090$

For the values of the last eight Schläfli symbols we see that the determinant of H is not greater than zero, so for these possibilities H is not positive definite as it should be and thus these possibilities are not allowed. As a result we find that the Schläfli symbols for a 5-dimensional regular polytope are

$$\{3,3,3,3\},\{4,3,3,3\},\{3,3,3,4\}$$

Extending these values even further, for $k>5$ dimensions we find the only possibilities to be

$$\{3,3,...,3,3\},\{4,3,...,3,3\},\{3,3,...,3,4\},\{4,3,...,3,4\}. \tag{4.18}$$

Using the following example we will see that $\{4,3,...,3,4\}$ is not an allowed possibility for any k. Take the basis $\{\vec{e}_1,...,\vec{e}_k\}$ for $\mathbb{E}^k$ with $e_i=(e_{i_1},...,e_{i_k})^\mathsf{T}$ and

$$e_{i_j}=\begin{cases}1, & \text{if } i=j\\ 0, & \text{if } i\neq j\ .\end{cases}$$

Clearly $\langle\vec{e}_i|\vec{e}_j\rangle=0$ if $i\neq j$. Consider the vectors $\vec{a}_1=\sqrt{2}\cdot\vec{e}_1$, $\vec{a}_i=\vec{e}_{i-1}+\vec{e}_i$ and $\vec{a}_{k+1}=\sqrt{2}\cdot\vec{e}_k$. For all $i=1,...,k+1$ the vector $\vec{a}_i$ has length $\sqrt{2}$, we have

$$\langle\vec{a}_i|\vec{a}_i\rangle=|\vec{a}_i|^2=2.$$

If $|i-j|>1$ then

$$\langle\vec{a}_i|\vec{a}_j\rangle=0.$$

Furthermore

$$\begin{aligned}\langle\vec{a}_1|\vec{a}_2\rangle&=\sqrt{2}\\ \langle\vec{a}_i|\vec{a}_{i+1}\rangle&=1 \text{ for i=2,...,k-1}\\ \langle\vec{a}_k|\vec{a}_{k+1}\rangle&=\sqrt{2}\end{aligned}$$

then for the angle θ_i formed by $\vec{a}_i,\vec{a}_{i+1}$ we find

$$\theta_i=\cos^{-1}\left(\frac{\langle\vec{a}_i|\vec{a}_{i+1}\rangle}{|\vec{a}_i|\cdot|\vec{a}_{i+1}|}\right)=\begin{cases}\frac{\pi}{4} & \text{if } i=1\\ \frac{\pi}{3} & \text{if } i=2,...,k-1\\ \frac{\pi}{4} & \text{if } i=k\ .\end{cases}$$

We now have enough information to create the following Gram matrix of the vectors $\vec{a}_i, i = 1, ..., k+1$.

$$G = \begin{pmatrix} 2 & 2\cdot\cos(\frac{\pi}{4}) & 0 & \dots & \dots & 0 \\ 2\cdot\cos(\frac{\pi}{4}) & 2 & 2\cdot\cos(\frac{\pi}{3}) & \ddots & & \vdots \\ 0 & 2\cdot\cos(\frac{\pi}{3}) & \ddots & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & 2\cdot\cos(\frac{\pi}{3}) & 0 \\ \vdots & & \ddots & 2\cdot\cos(\frac{\pi}{3}) & 2 & 2\cdot\cos(\frac{\pi}{4}) \\ 0 & \dots & \dots & 0 & 2\cdot\cos(\frac{\pi}{4}) & 2 \end{pmatrix} \tag{4.19}$$

As one can see, this is a Gram matrix corresponding to the Schläfli symbol $\{4, 3, ..., 3, 4\}$ of length k. The vectors $\vec{a}_i$ for $i = 1, ..., k+1$ are $k+1$ vectors in the k-dimensional space $\mathbb{E}^k$ and thus are linearly dependent. Therefore the rank of the matrix A with row vectors $\vec{a}_i$ is at most k, $rank(A) \geq k$. For the Gram matrix it holds $rank(G) = rank(A) \geq k$. So G does not have full rank. If a matrix is not of full rank then it does not have an inverse and is therefore singular. So G is singular, it then follows $det(G) = 0$.

To conclude, for every k the Gram matrix G corresponding to the Schläfli symbol $\{4, 3, ..., 3, 4\}$ of length k has $det(G) = 0$. As we have already discussed the determinant of a Gram matrix corresponding to the normal vectors of the hyperplanes with a diagram from theorem 4.2.1 should be greater than zero. So $\{4, 3, ..., 3, 4\}$ does not meet the requirements.
Thus for $n > 5$ there exist only the Schläfli symbols

$$\{3, 3, ..., 3, 3\}, \{4, 3, ..., 3, 3\}, \{3, 3, ..., 3, 4\}$$

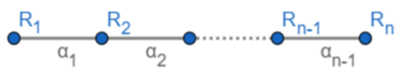
for an n-dimensional polytope. □

4.2.2 Wythoff's construction

We have determined the Schläfli symbols in different dimensions. The following theorem will prove that all these Schläfli symbols actually define regular polytopes up to a regular dual polytope.

> **Theorem 4.2.6.** (Wythoff construction) If there exist reflections $R_1, ..., R_n$ with a diagram mentioned by theorem 4.2.3, then there exists an n-dimensional regular polytope, up to isomorphisms, with the symmetry group $\langle R_1, ..., R_n\rangle$.

Proof. Suppose there exist reflections $R_1, ..., R_n$ with diagram:



and where the α_i's have the values of a Schläfli symbol mentioned in theorem 4.2.3.

We will first construct a flag from these reflections. Let G be the group of symmetries generated by the reflections $R_1, ..., R_n$, $G = \langle R_1, ..., R_n \rangle$. Let r_i denote the hyperplane left invariant by the reflection R_i for all $i \in \{1, ..., n\}$, i.e. r_i is the reflection axis of R_i. Choose a point $p \in r_2 \cap r_3 \cap ... \cap r_n$, this is our initial vertex. Let V be a vertex set, it is the set of points $v = \sigma(p)$ where $\sigma \in G$.

We set $C_1 = \{p\}$ and for $j \in \{2, \dots, n\}$ we recursively define [8]

$$C_j := C_j := \{\langle R_1, ..., R_{j-1} \rangle v : v \in C_{j-1}\} \tag{4.20}$$

This means C_j is the set of images under the symmetry group generated by the reflections $R_1, \dots, R_{j-1}$ of C_{j-1}. So $C_1 = \{p\}$, $C_2 = \{p, R_1 p\}$, etc.

This results in the maximal base flag $F := (C_1, \dots, C_n)$. Let the polytope $\mathcal{F}$ be the set of all flags $\sigma(F)$ with $\sigma \in G$, $\mathcal{F} := \{\sigma(F) : \sigma \in G\}$. This by definition is the orbit of F and is therefore a transitive action.
Secondly we need to prove the following two lemmas.

Lemma 4.2.7. The maximal flag $F := (C_1, \dots, C_n)$ has a unique adjacent flag F_i for every $i \in \{1, ..., n\}$ such that F and F_i only differ by their i-cells and the reflection R_i reflects F into F_i.

Proof. Let $F := (C_1, \dots, C_n)$ be the maximal base flag, by the diamond property 2.3.4 it holds that for every $i \in \{1, ..., n\}$ there exist exactly two i-cells C_i, C_i' such that $C_{i-1} \subseteq C_i, C_i' \subseteq C_{i+1}$. Thus for every i the elements of the sequence of j-cells $(C_1, ..., C_{i-1}, C_i', C_{i+1}, ..., C_n)$ are also mutually incident and thus $F_i = (C_1, ..., C_{i-1}, C_i', C_{i+1}, ..., C_n)$ is a unique maximal flag that differs from F only by its i-cell, F_i is a unique adjacent flag of F. □

By transitivity of G on the maximal flags of Π it does not matter which starting flag we take.

Lemma 4.2.8. If $\mathcal{F} := \{\sigma(F) : \sigma \in G\}$ then $\mathcal{F}$ is connected on its flags. i.e., given two flags $F, g(F) \in \mathcal{F}$, there exists a sequence of flags $(F = F_1, F_2, ..., F_k = g(F))$ having F_i adjacent to F_{i+1} for all i.

Proof. Consider our base flag F and let $g \in G = \langle R_1, \dots, R_n \rangle$ then $g(F) \in \mathcal{F}$ is an image of our base flag. Suppose the length of g is the minimal number of reflections R_i needed to construct the symmetry g.
Base case: If the length of g is 0 then $F = g(F)$. If the length of g is 1 then $g = R_j$ for some $j \in \{1, ..., n\}$ which means F and $g(F)$ are adjacent flags. In this case there exists a sequence of flags $(F = F_1, F_2, ..., F_k = g(F)$ having F_i adjacent to F_{i+1} for all i, namely the sequence $(F = F_1, F_2 = g(F)$.
Induction hypothesis: Let $g = R_{i_1} \cdots R_{i_k}$ of length $k > 1$ and suppose there exists a sequence of consecutively adjacent flags from F to $F' = R_{i_2} \cdots R_{i_k} F$, namely $(F, R_{i_k} F, R_{i_{k-1}} R_{i_k} F, ..., F')$.
Induction step: We know that $R_1 F$ is an adjacent flag of F so there exists the sequence of consecutively adjacent flags $(R_1 F, F)$. Then by our induction hypothesis we know that there exists a sequence of consecutively adjacent flags from $R_1 F$ to F', namely $(R_1 F, F, R_{i_k} F, R_{i_{k-1}} R_{i_k} F, ..., F')$. But then there also exists a sequence of consecutively adjacent flags from $R_1 R_1 F$ to $R_1 F'$, we have $(R_1 R_F F = F, F, R_{i_k} F, R_{i_{k-1}} R_{i_k} F, ..., R_1 F' = g(F))$.
Thus our base flag F and its image $g(F)$ are connected through some sequence of flags $(F = F_1, F_2, ..., F_k = g(F)$ having F_i adjacent to F_{i+1} for all i. □

In conclusion, we have established that a base flag can be constructed from the given reflections $R_1, ..., R_n$, that each flag has n unique adjacent flags, that $\mathcal{F}$ is connected on its flags and also that $G = \langle R_1, ..., R_n \rangle$ is transitive on the maximal flags of $\mathcal{F}$. Therefore, if there exist reflections $R_1, ..., R_n$ with a diagram from theorem 4.2.1 then $\mathcal{F}$ is an n-dimensional regular polytope with its symmetry group $\langle R_1, ..., R_n \rangle$. □

We now know that the symmetry group of a polytope is uniquely defined by a diagram. Such diagrams can be described by a Schläfli symbol and we have found that there is a limited number of Schläfli symbols corresponding to such diagrams. We have also proved that there exists a way to construct a regular polytope from a set of reflections that correspond to some Schläfli symbol. Thus we have found the number of regular polytopes existing in n dimensions for each $n \geq 2$. We will be using this knowledge in chapter 5 to construct and create drawings for some examples of regular polytopes.

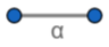
Chapter 5

Constructing regular polytopes

In this chapter we will first see two examples of how to use Wythoff's construction to actually construct regular polygons. Then we will discuss the algorithm used to find construct higher dimensional polytopes.

5.1 Polygon

To construct a polygon using Wythoff's construction we start with two reflections belonging to the diagram:



Let us use the instance where $\alpha = 7$ as an example. We will do two separate constructions. We will use the two lines (hyperplanes) r_1, r_2, in the first case they are inclined at an angle $\frac{\pi}{7}$ and in the second case at an angle $\frac{3\pi}{7}$. R_1, R_2 are the reflections in the hyperplanes r_1, r_2 respectively. Let r_1 in both cases be the vertical line. Figures 5.1a, 5.2a show a chosen starting vertex v on r_2. We reflect v by R_1 to get $R_1(v)$ and find the edge $e = (v, R_1(v))$. (v, e) is a maximal flag of a polygon. The edge is depicted in the figures 5.1b, 5.2b.

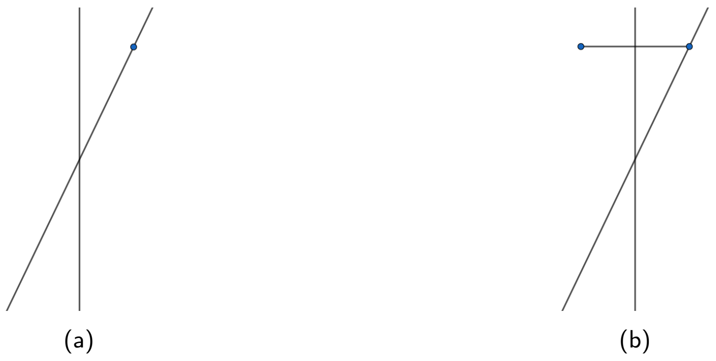
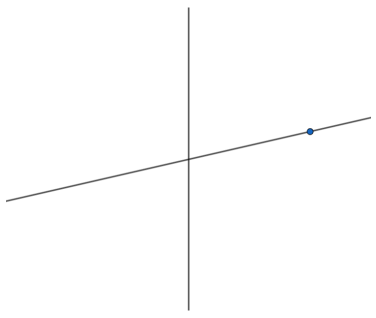


Figure 5.1: Constructing a regular 7-gon, case 1; part 1/3


(a)

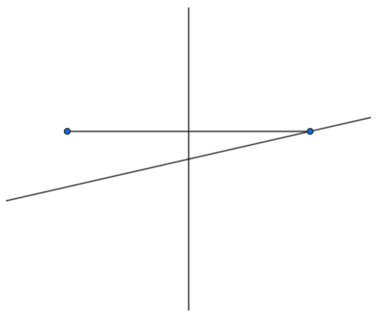

(b)

Figure 5.2: Constructing a regular 7-gon, case 2; part 1/3

As we now have a maximal flag of the polygon, we can find all maximal flags by successively applying the two reflections R_1, R_2. We first reflect e with R_2 to get $R_2(e)$, then we reflect this result with R_1 to get $R_1R_2(e)$ etc.
This process of alternately applying R_1 and R_2 to the former reflection image is depicted in the figures 5.3a, 5.3b, 5.3c for the first case and in 5.4a, 5.4b, 5.4c for the second.

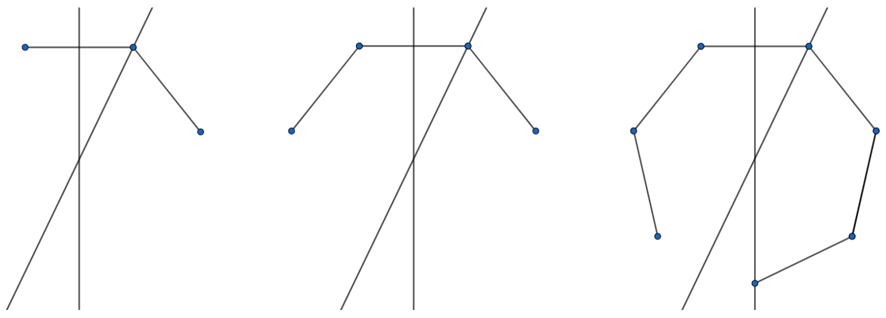

(a) (b) (c)

Figure 5.3: Constructing a regular 7-gon, case 1; part 2/3

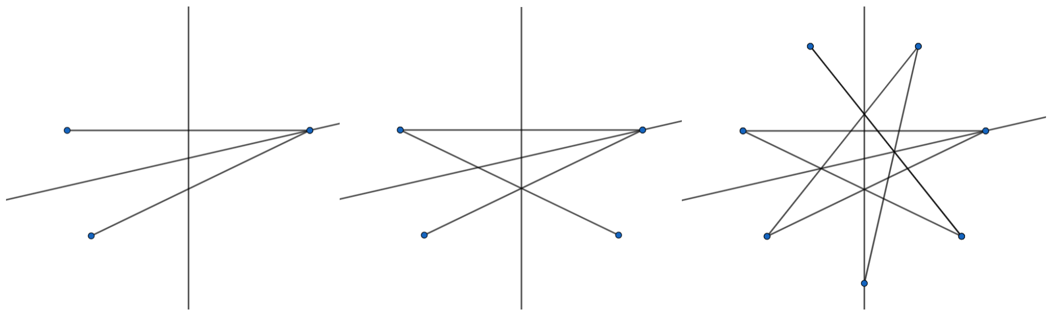
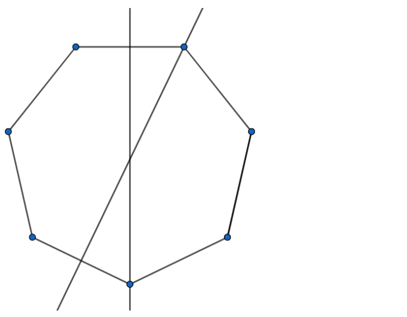

(a) (b) (c)

Figure 5.4: Constructing a regular 7-gon, case 2; part 2/3

Once this process does not find any new edges then we stop and have found the full polygon. In this example we have found the heptagon and a stellated regular 7-gon, the results are depicted in the figures 5.5a, 5.5b.


(a) Constructing a regular 7-gon, case 1; part 3/3

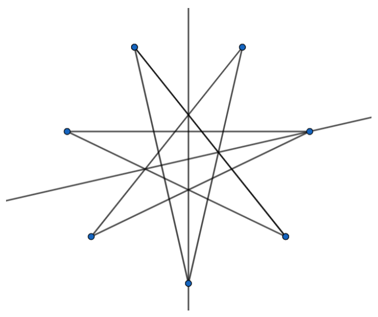

(b) Constructing a regular 7-gon, case 2; part 3/3

Figure 5.5

5.2 Higher dimensional polytopes

Let $n > 2$. We describe an algorithm to construct n-dimensional polytopes using Wythoff's construction.
We will discuss the algorithm using pseudo-code and use the example of the dodecahedron to have a better understanding of what the algorithm does.

5.2.1 Initialisation

Input: A Schläfli symbol $\{p_1, p_2, ..., p_{n-1}\}$ and a node j of the diagram corresponding to this Schläfli symbol.
Determine n affine hyperplanes r_i for $i = 1, ..., n$ with corresponding unitary normal vectors $\vec{a}_i$ satisfying:

$$\langle \vec{a}_i | \vec{a}_j \rangle = 0 \text{ for } i > j+1 \text{ or } j > i+1 \tag{5.1}$$

$$\langle \vec{a}_i | \vec{a}_{i+1} \rangle = \cos\left(\frac{k\pi}{p_i}\right) \text{ for } i = 1, ..., n-1 \text{ and } k = 1, 2, ... \tag{5.2}$$

Define $R_i(\vec{v}) = \vec{v} - 2\langle \vec{v} | \vec{a}_i \rangle \vec{a}_i$ to be the reflection in r_i for $i = 1, ..., n$.

5.2.2 Flag construction

After the initialisation we apply the following algorithm to find a maximal flag of the polytope.

```
Algorithm 1: Flag construction
Result: Maximal flag $F = (C_1, C_2, ..., C_n)$ or flags
        $F_1 = (C_1, C_2, ..., C_j), F_2 = (D_1, D_2, ..., D_{n-j+1})$
⋮
initialisation:
Determine: a vertex $C_1 = v_1 \in \bigcap_{i\in\{1,...,n:i\neq j\}} r_i$ and $v_1 \neq 0$.
if $j = 1$ or $j = n$ ;           /* node $j$ is at one of the ends of a diagram */
 then
  $F = (C_1)$
else
  $F_1 = F_2 = (C_1)$ ;    /* node $j$ is not an end-node and we can go left or right
   through the diagram */
end
⋮
construction:
if $j = 1$ then
  $i = 2$ ;                        /* We are on the most left node of the diagram */
  while $F$ contains less than $n$ elements do
    $C_i := \{\langle R_1, R_2, ..., R_{i-1}\rangle v : v \in C_{i-1}\}$
    $F \leftarrow$Append$(C_i, F)$ such that $C_i$ is the last element of $F$.
    $i \leftarrow i + 1$
  end
else if $j = n$ then
  $i = 2$ ;                       /* We are on the most right node of the diagram */
  while $F$ contains less than $n$ elements do
    $C_i := \{\langle R_n, R_{n-1}, ..., R_{n-i+1}\rangle v : v \in C_{i-1}\}$
    $F \leftarrow$Append$(C_i, F)$ such that $C_i$ is the last element of $F$.
    $i \leftarrow i + 1$
  end
else
  $i = 2$ ;                           /* We are not on an end-node of the diagram */
  $k = 2$
  while $j - i \geq 0$ do
    $C_i := \{\langle R_j, R_{j-1}, ..., R_{j-i+1}\rangle v : v \in C_{i-1}\}$ ;     /* We go to the left in the
     diagram */
    $F_1 \leftarrow$Append$(C_i, F)$ such that $C_i$ is the last element of $F_1$.
    $i \leftarrow i + 1$
  end
  while $j + k \leq n$ do
    $D_k := \{\langle R_j, R_{j+1}, ..., R_{j+k-1}\rangle v : v \in D_{k-1}\}$ ;    /* We go to the right in the
     diagram */
    $F_2 \leftarrow$Append$(D_k, F)$ such that $D_k$ is the last element of $F_2$.
    $k \leftarrow k + 1$
  end
end
```

5.2.3 Building the polytope

Once we have our initial maximal flag(s) we can apply the symmetry group of the polytope to this flag to find all maximal flags of the polytope, the union of these flags is the polytope. This is done using the following algorithm.

Algorithm 2: Building the polytope

```
Result: Polytope $\Pi$
⋮
initialisation:
if $j = 1$ or $j = n$ then
    $\Pi = \{F\}$
else
    $\Pi = \{F_1, F_2\}$
end
⋮
construction:
$i = 1$
$N = 1$
while $i \leq N$ do
    $F_i = \Pi[i]$ ;                     /* $\Pi[i]$ is the $i^{th}$ element of the set $\Pi$ */
    for $j = 1$ to $n$ do
        if $R_j F_i \notin \Pi$ then
            $\Pi \leftarrow \Pi \cup R_j F_i$
        end
    end
    $i \leftarrow i + 1$
    $N = \text{Length}(\Pi)$ ;      /* Length($\Pi$) is the current number of elements in $\Pi$ */
end
```

5.2.4 Constructing the Dodecahedron

In this section we see an example of the application of the algorithm discussed in this chapter.

Initialisation

We use the input: $\{5, 3\}$, $j = 1$.
And from this we compute the following normal vectors:

$$\vec{a}_1 = (1, 0, 0)^\intercal$$

$$\vec{a}_2 = (\frac{1}{4}(1 + \sqrt{5}), \frac{1}{2}\sqrt{\frac{1}{2}(5 - \sqrt{5})}, 0)^\intercal$$

$$\vec{a}_3 = (0, \sqrt{\frac{1}{10}(5 + \sqrt{5})}, \sqrt{\frac{1}{10}(5 - \sqrt{5})})^\intercal$$

With reflections:

$$R_1(\vec{v}) = \vec{v} - 2\langle \vec{v} | \vec{a}_1 \rangle \vec{a}_1$$
$$R_2(\vec{v}) = \vec{v} - 2\langle \vec{v} | \vec{a}_2 \rangle \vec{a}_2$$
$$R_3(\vec{v}) = \vec{v} - 2\langle \vec{v} | \vec{a}_3 \rangle \vec{a}_3$$

Flag construction

We determine the vertex v_1 that is contained in the intersection of the affine hyperplanes r_2, r_3:

$$v_1 = (1, -\sqrt{1 + \frac{2}{\sqrt{5}}}, \sqrt{\frac{5}{2} + \frac{11}{2\sqrt{5}}})^\intercal.$$

In figure 5.6 the first element $C_1 = v_1$ of a maximal flag F is shown.

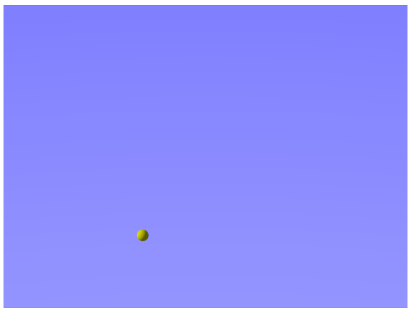


Figure 5.6

We find the vertex v_2 by applying the reflection R_1 to v_1, we get:

$$v_2 = (-1, -\sqrt{1 + \frac{2}{\sqrt{5}}}, \sqrt{\frac{5}{2} + \frac{11}{2\sqrt{5}}})^\intercal$$

and thus find a second element $C_2 = \{v_1, v_2\}$ of F.
This is depicted in figure 5.7.

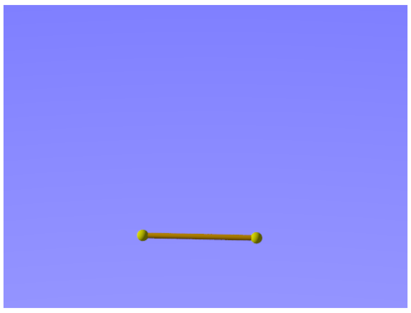


Figure 5.7

Then by applying symmetries from the symmetry group generated by the reflections R_1, R_2 we construct the third element C_3 of the polyhedron. An interim result and end result of the construction

of C_3 are shown in figure 5.8a and 5.8b.
The figure shown in figure 5.8b is a maximal flag of the polyhedron.

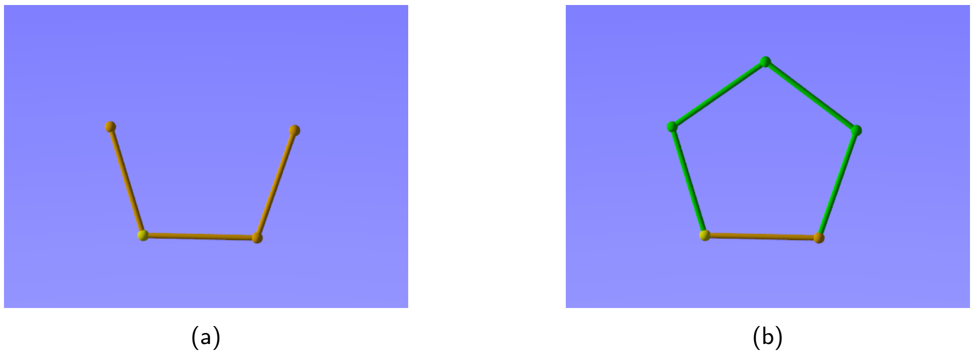

(a) (b)

Figure 5.8

Building the polytope

After constructing a maximal flag of the polyhedron we will build the full polyhedron by computing all images of the constructed maximal flag under symmetries from the group $\langle R_1, R_2, R_3 \rangle$. Figures 5.9a and 5.9b give intermediate results of this construction and figure 5.9c provides an image of the full Dodecahedron.

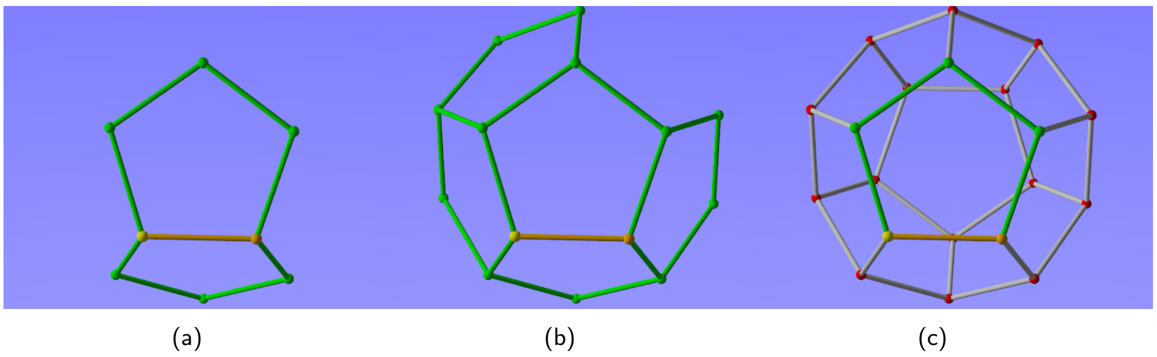

(a) (b) (c)

Figure 5.9

Chapter 6

4D polytopes

We have implemented the algorithm of chapter 5.2.1 in Mathematica [12]. The resulting polytope is used as input in POV-Ray [1]. POV-Ray stands for Persistence of Vision Raytracer and is a high-quality software tool for creating stunning three-dimensional graphics.
In this chapter we will depict the illustrations of the 3-and 4-dimensional regular polytopes. In section 6.2 and 6.3 we discuss two polytopes corresponding to the Schläfli symbols $\{4,3,3\},\{3,3,3\}$ in detail with multiple figures that correspond to some of the details known for these polytopes. In section 6.4 a table containing all 3- and 4-dimensional diagrams and corresponding polytope representations is given.

6.1 Realisation of the figures

We use Mathematica with the algorithm discussed in chapter 5.2.1 to find the coordinates of the vertices and to find the compositions of the edges, faces and cells of a 4-dimensional polytope.
However before we can realise a representation of a 4-dimensional polytope with the use of POV-Ray we need to translate the 4-dimensional coordinates into 3-dimensional ones that can serve as input for POV-Ray.
This is done via an orthogonal projection of the coordinates onto an affine hyperplane of the 4-dimensional space. Let $\vec{u}$ be a unitary normal vector of the hyperplane and let $\{\vec{w}_1,\vec{w}_2,\vec{w}_3\}$ be an orthonormal basis of the hyperplane. Furthermore let $\vec{t}$ be the translation vector of the hyperplane in case the origin is not contained in the hyperplane.
We project the vertices of the polytope using the following function for the projection image $P(\vec{v})$ of a vector $\vec{v}$:

$$P(\vec{v}) = \vec{v} - \langle \vec{v}|\vec{u}\rangle \cdot \vec{u} + \langle \vec{t}|\vec{u}\rangle \cdot \vec{u}. \tag{6.1}$$

Let $\vec{x}$ be the solution to $W\vec{x} = P(\vec{v})$ where W is the 3×4-matrix with row vectors w_1, w_2, w_3. We consider the coordinates of the vector $\vec{x}$ to be the 3-dimensional coordinates of the vector $\vec{v}$.

Now that we have established 3-dimensional coordinates for the vertices we are able to use the program POV-Ray for making a representation of the polytope.

6.2 {4,3,3}

In figure 6.1 a representation of a polytope corresponding to the Schlafli symbol $\{4,3,3\}$ and starting on the first node of the diagram is shown. Such a polytope is also called a tesseract.

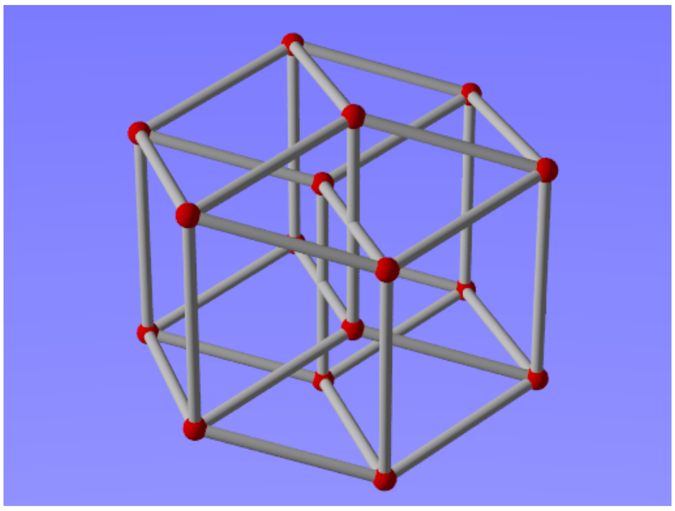


Figure 6.1: 3-dimensional representation of the tesseract

Figure 6.2 shows two maximal flags of the polytope:

- a vertex colored yellow;
- an edge colored orange;
- a face colored green;
- and two cells (blue and red) incident to the green face.

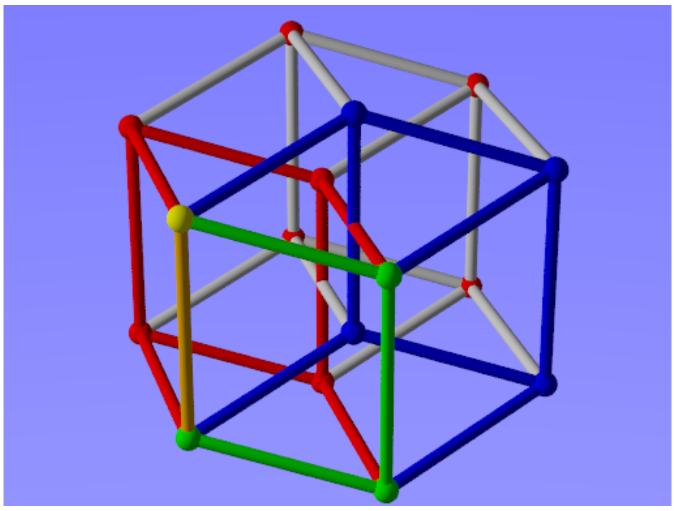


Figure 6.2: 3-dimensional representation of the tesseract

The cells of the tesseract are cubes. One of such cells is depicted in figure 6.3.

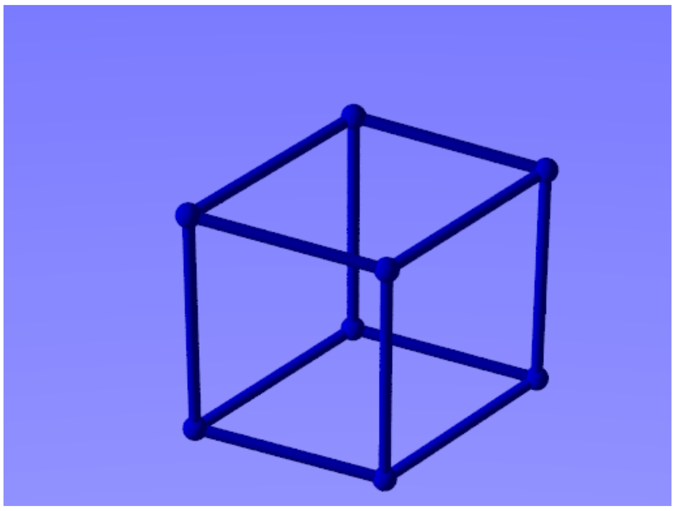


Figure 6.3

For the tesseract we have the following table of details:

Number of vertices in the polytope	16
Number of edges in the polytope	32
Number of vertices in a face	4
Number of faces in the polytope	24
Number of vertices in a cell	8
Number of cells in the polytope	8
Number of edges incident to each vertex	4
Number of faces incident to each vertex	6
Number of cells incident to each vertex	4
Number of faces incident to each edge	3
Number of cells incident to each face	2

In figure 6.4 the 4 cells incident to one vertex are shown and figure 6.5 shows the 6 faces incident to one vertex.

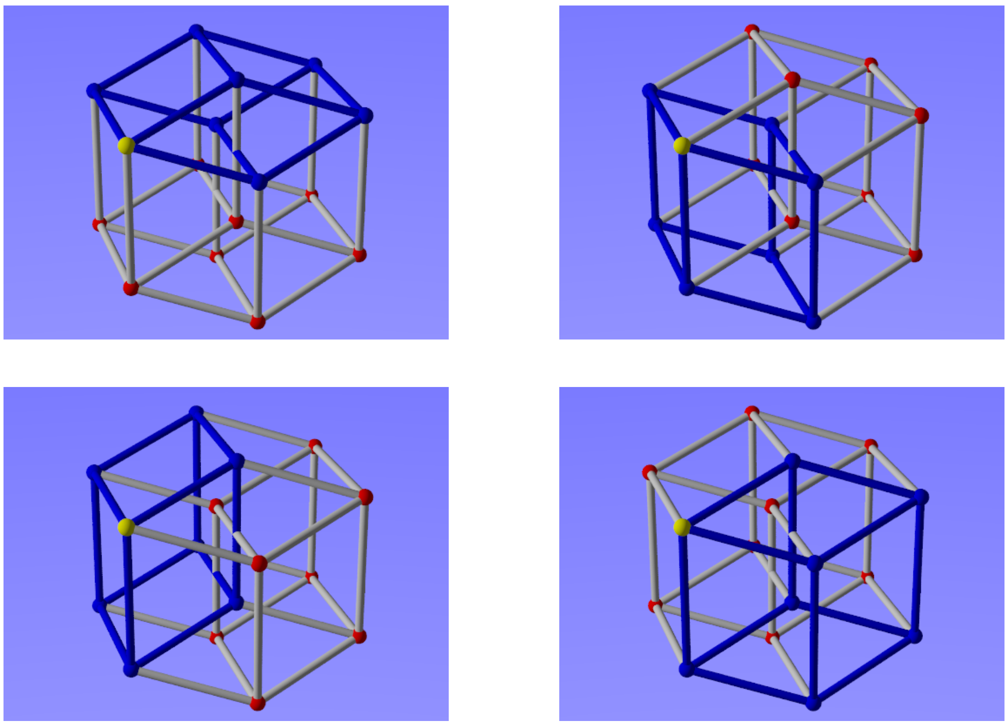


Figure 6.4: The 4 cells incident to one vertex.

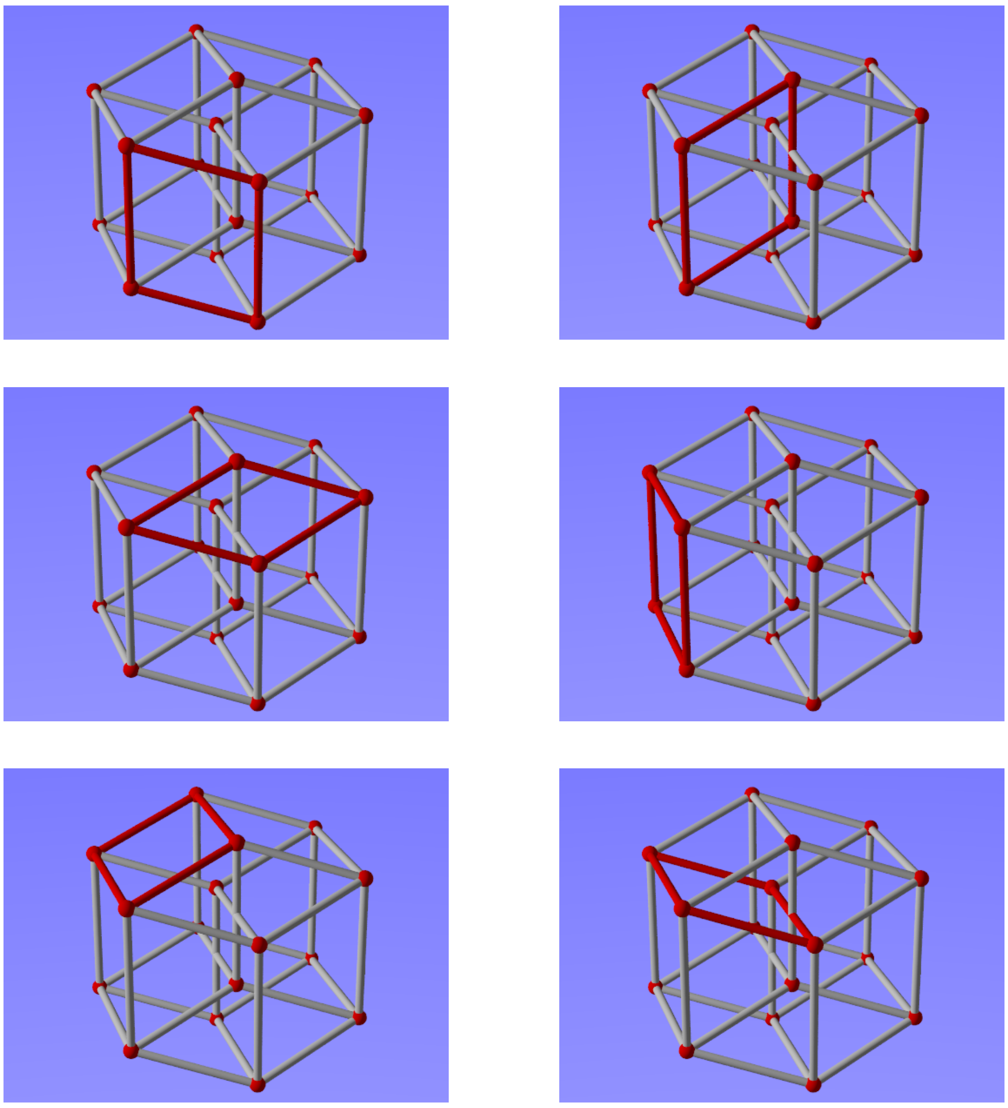


Figure 6.5: The 4 cells incident to one vertex.

6.3 {3,3,3}

In figure 6.6 a representation of a polytope corresponding to the Schlafli symbol $\{3,3,3\}$ and starting on the first node of the diagram is shown. Such a polytope is also called a tesseract.

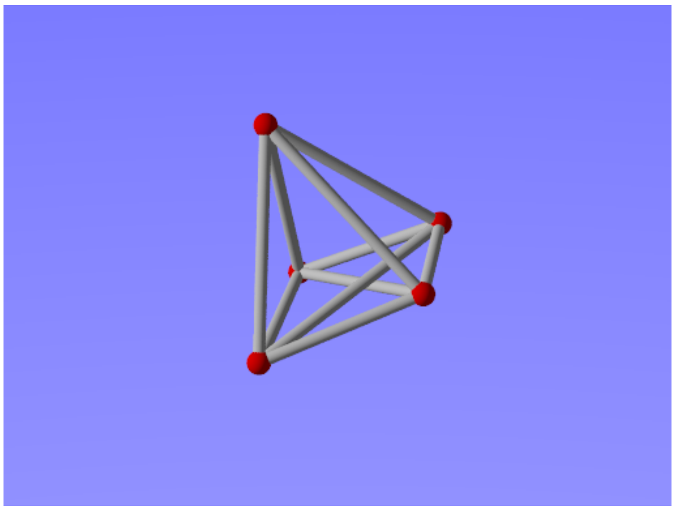


Figure 6.6: 3-dimensional representation of a polytope corresponding to $\{3, 3, 3\}$

Figure 6.7 shows two maximal flags of the polytope:

- a vertex colored yellow;
- an edge colored orange;
- a face colored green;
- and a cell colored blue.

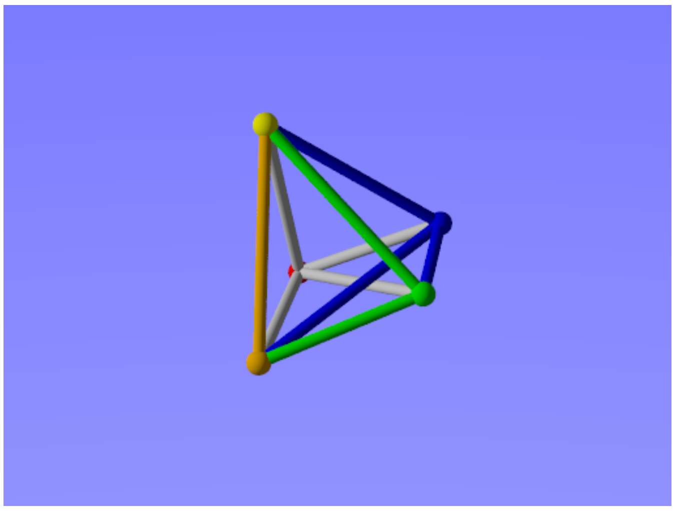


Figure 6.7: 3-dimensional representation of a polytope corresponding to $\{3, 3, 3\}$

The cells of this polytope are tetrahedra. One of such cells is depicted in figure 6.8.

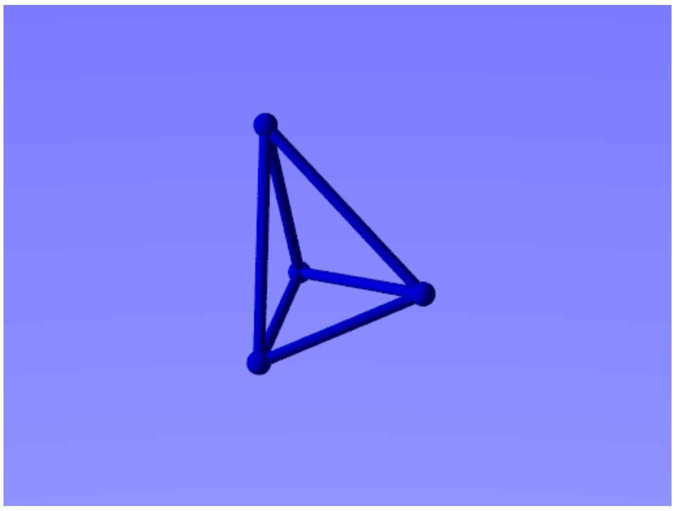


Figure 6.8

For the polytope we have the following table of details:

Number of vertices in the polytope	5
Number of edges in the polytope	10
Number of vertices in a face	3
Number of faces in the polytope	10
Number of vertices in a cell	4
Number of cells in the polytope	5
Number of edges incident to each vertex	4
Number of faces incident to each vertex	6
Number of cells incident to each vertex	4
Number of faces incident to each edge	3
Number of cells incident to each face	2

In figure 6.9 the 4 cells incident to one vertex are shown and figure 6.10 shows the 6 faces incident to one vertex.

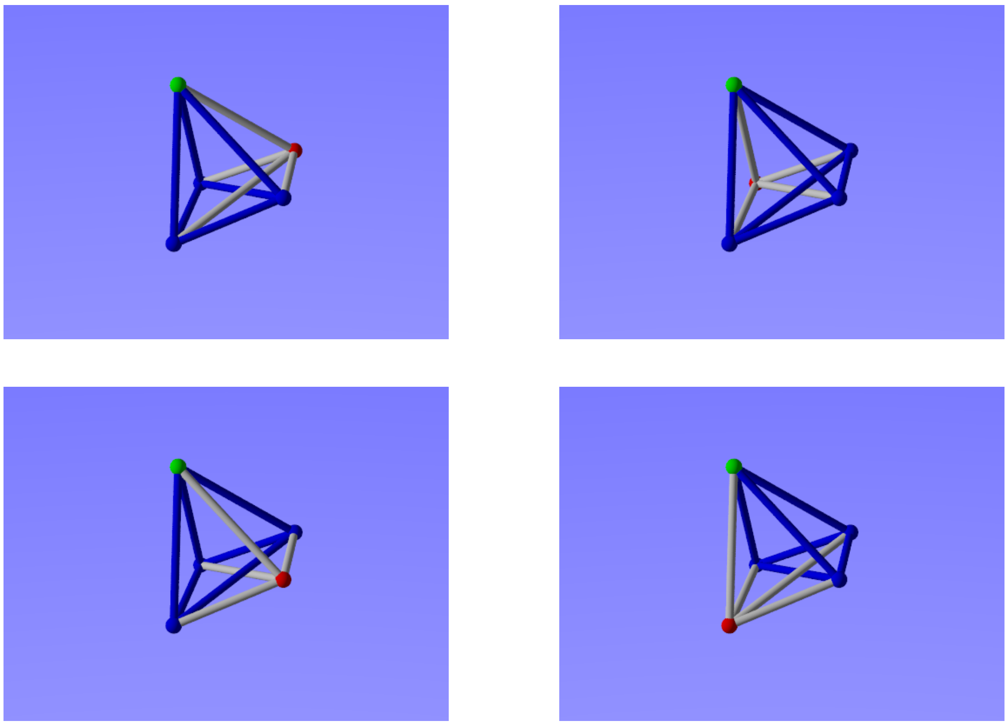


Figure 6.9: The 4 cells incident to one vertex.

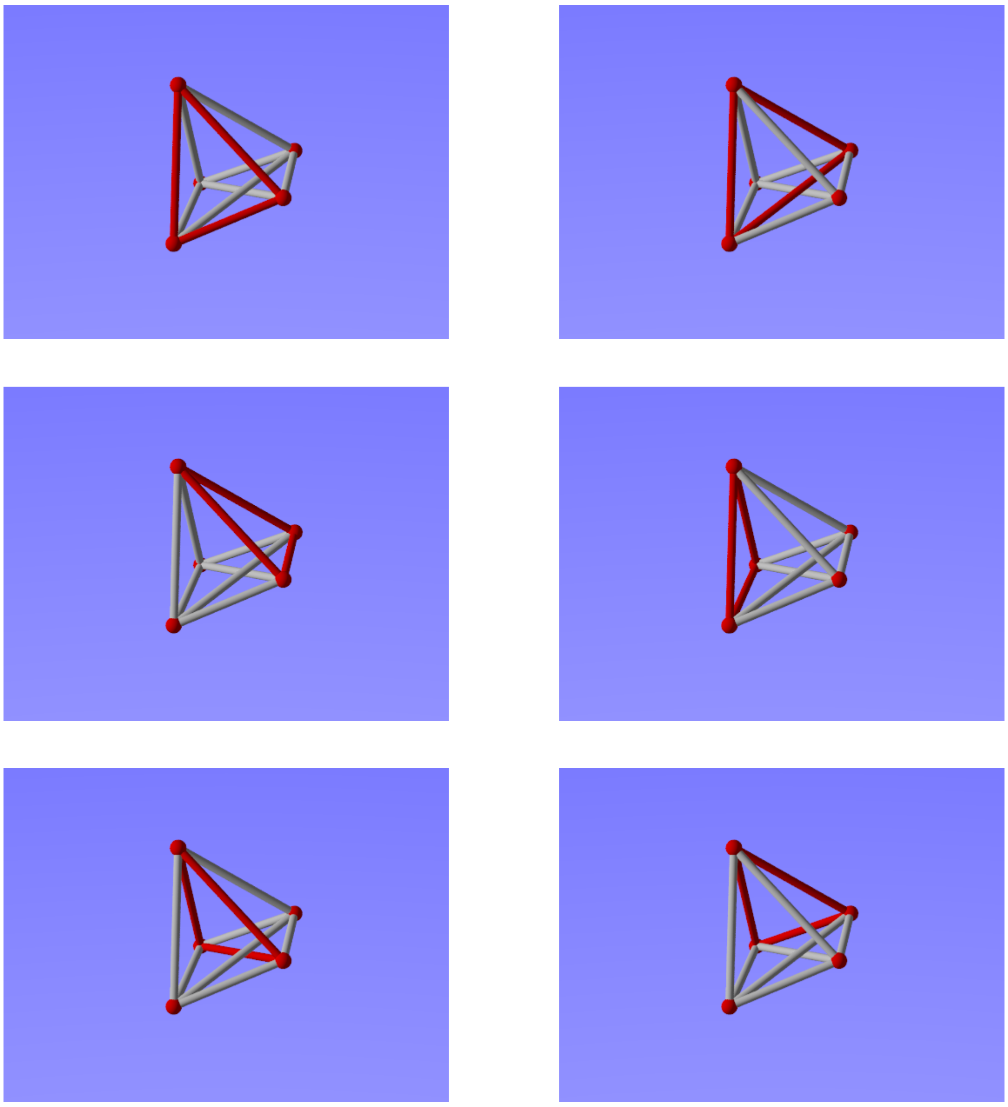
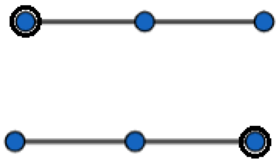
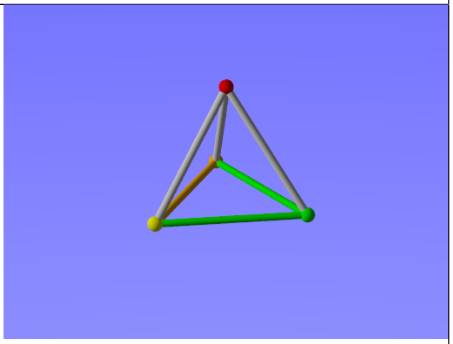
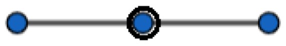
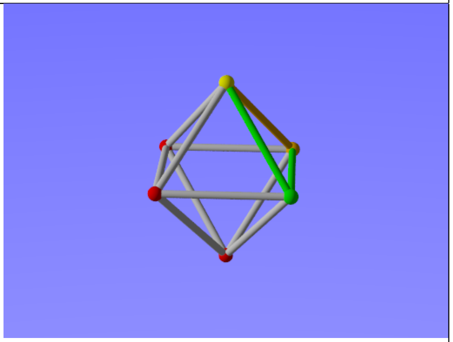
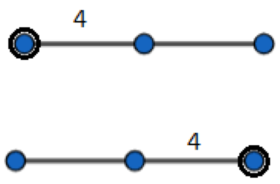
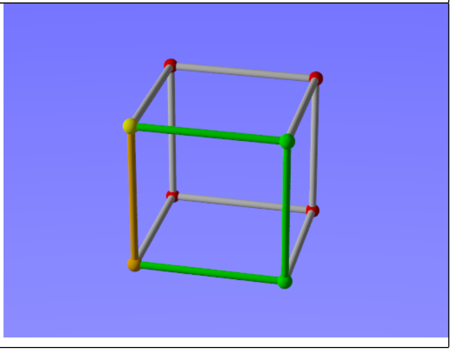
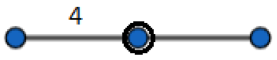
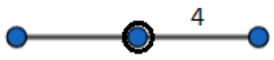
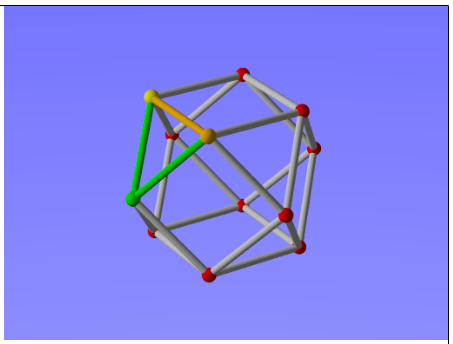
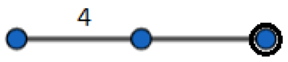
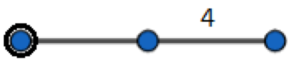
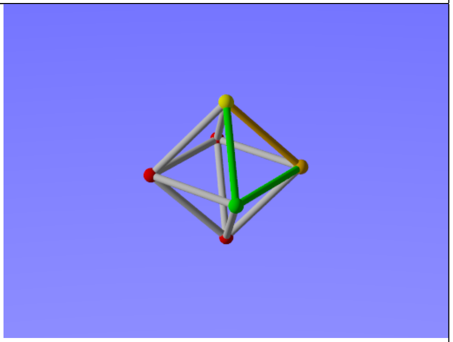
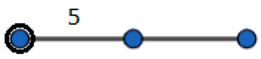
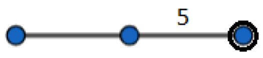
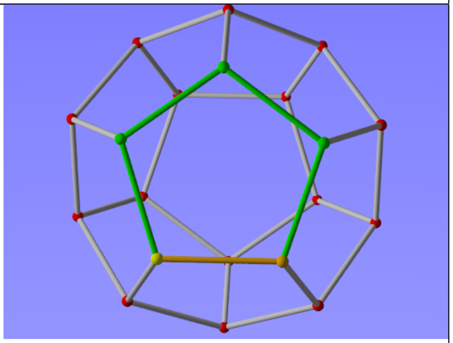
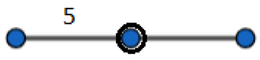
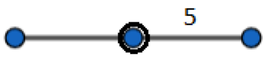
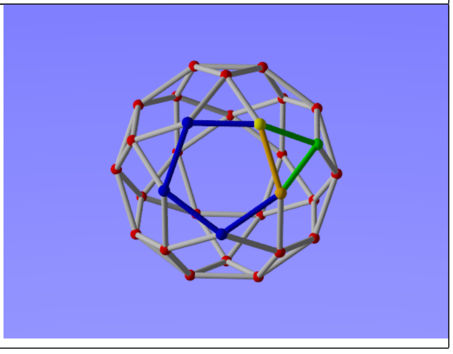


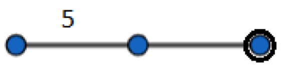
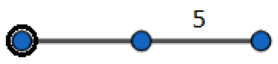
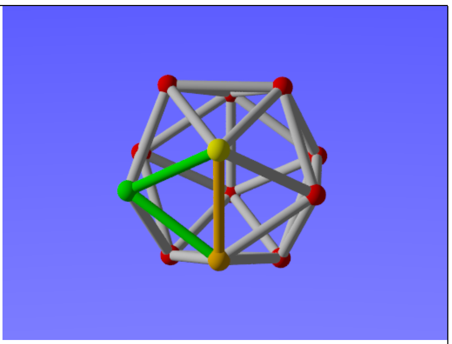
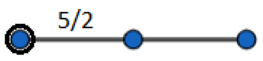
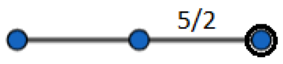
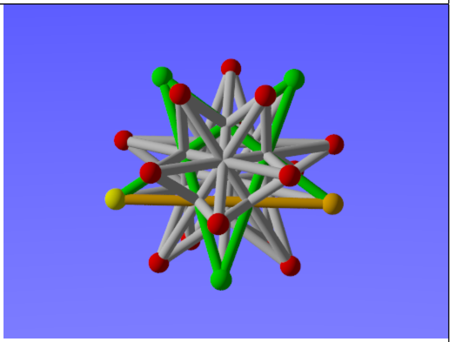
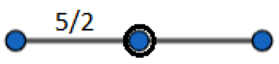
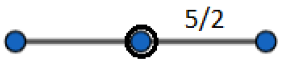
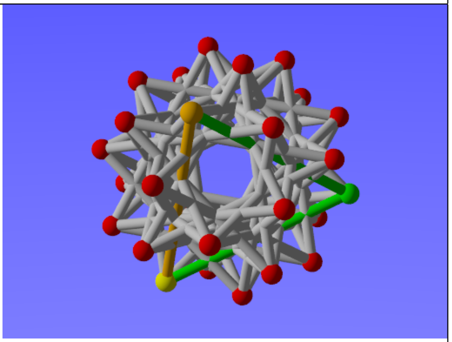
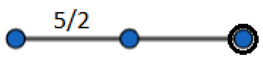
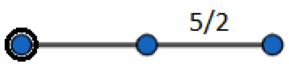
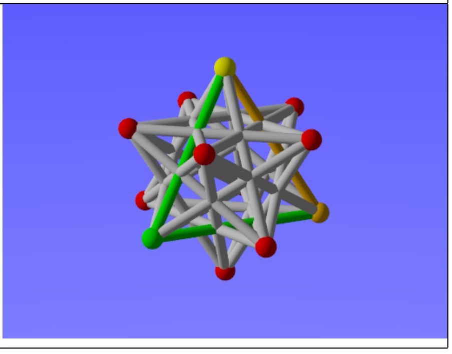
Figure 6.10: The 4 cells incident to one vertex.

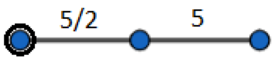
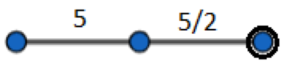
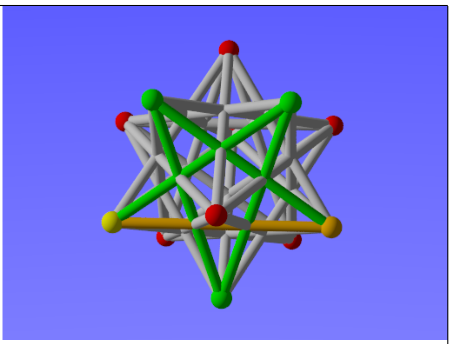
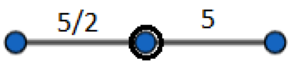
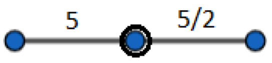
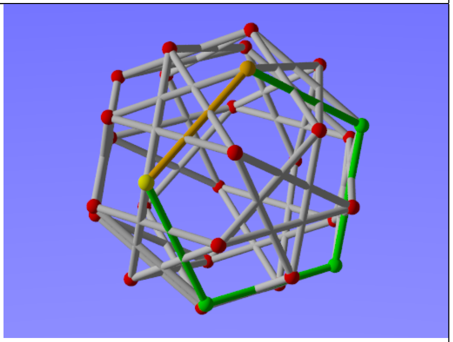
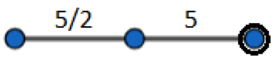
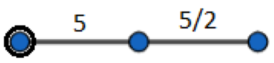
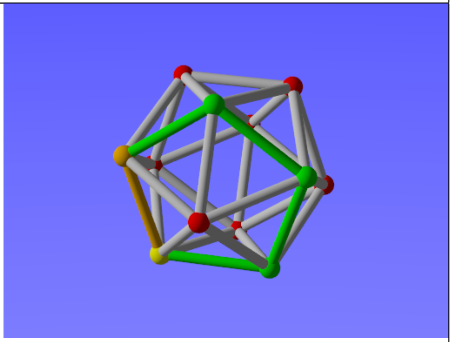
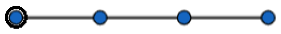
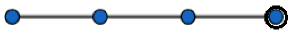
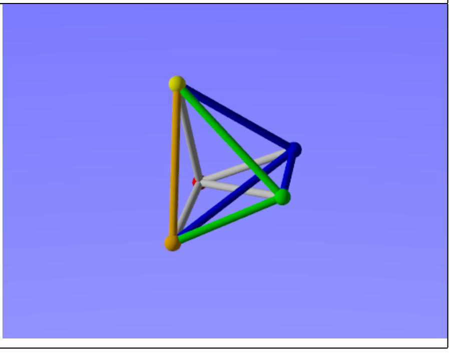
6.4 More polytopes

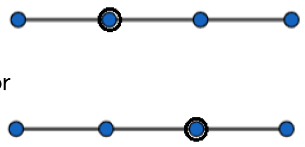
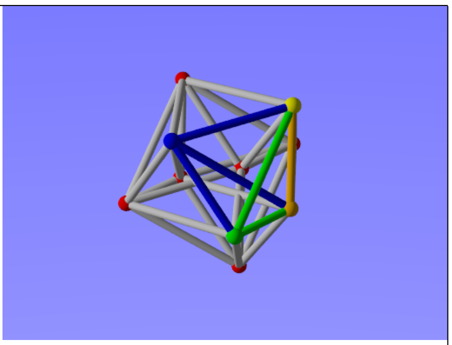
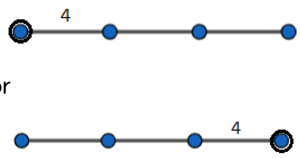
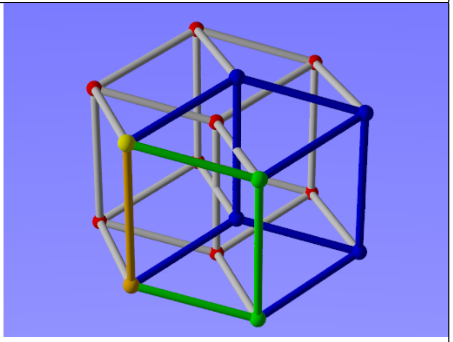
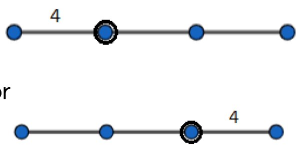
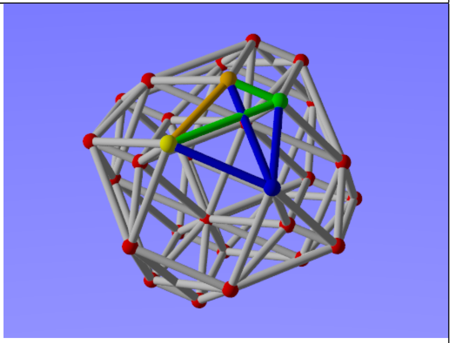
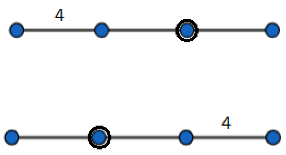
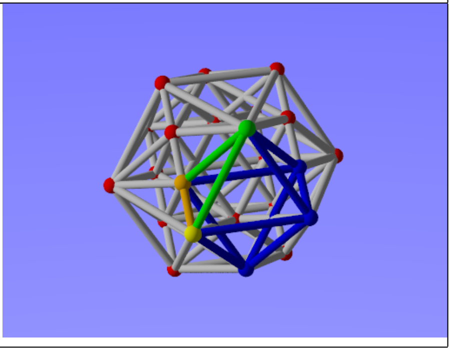
Recall: in the diagrams that correspond to Schläfli symbols, the value 3 occurs frequently and is therefore omitted.
In the table below all diagrams of the 3- and 4-dimensional regular polytopes are given with their corresponding 3-dimensional representation. The numbers in the first column of the table are reference numbers and refer to the diagram and polytope on that row. After the table we will discuss some relations that exist between the polytope representations.
We also have representations of stellated polytopes in 3- and 4-dimensional space. For these polytopes the corresponding diagrams in the table are adjusted to indicate the correct angle between the reflection axes of the generating reflections. So instead of the order 5 of two reflections, these diagrams will mention $5/2$ to refer to the angle $\frac{2\pi}{5}$.

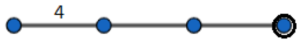
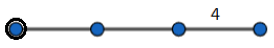
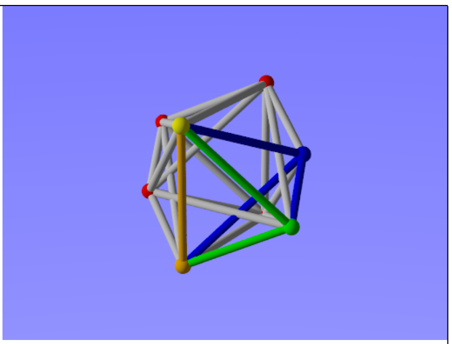
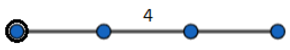
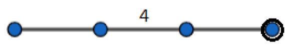
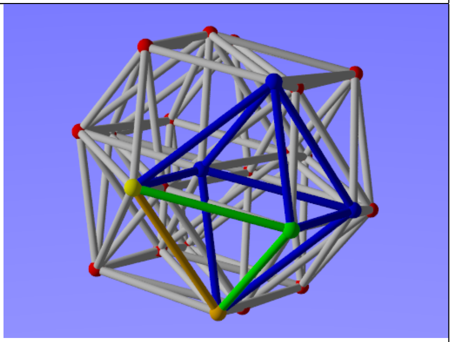
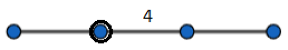
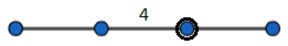
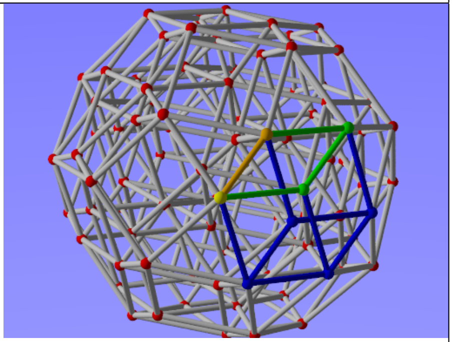
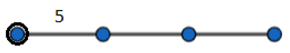
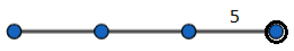
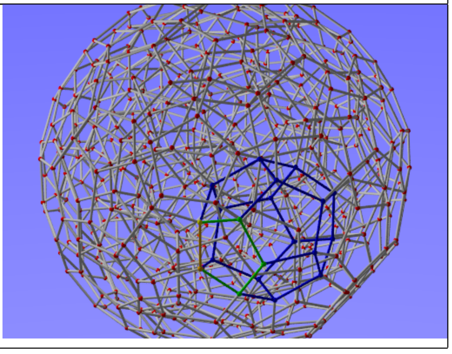
Nr.	Diagram	Corresponding representation
	3D diagrams:	
1		
2		
3		

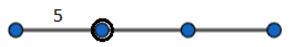
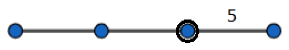
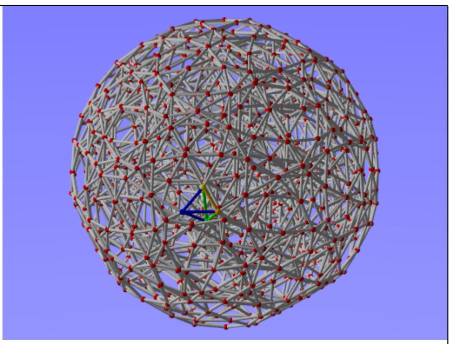
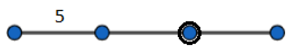
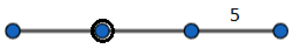
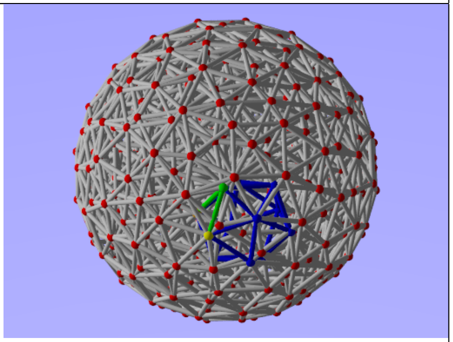
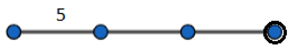
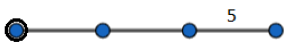
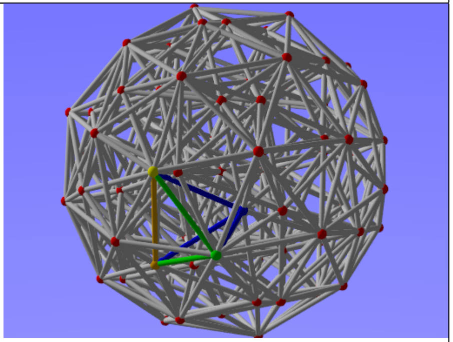
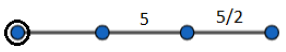
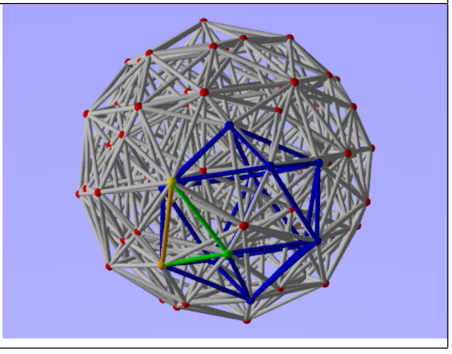
4	 or 	
5	 or 	
6	 or 	
7	 or 	

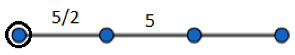
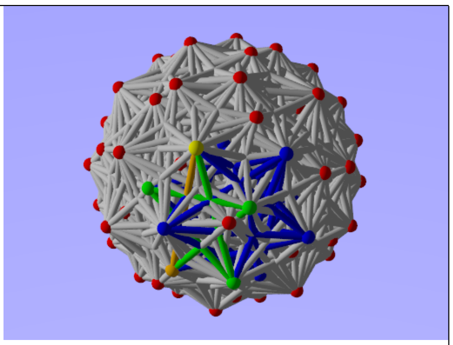
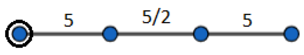
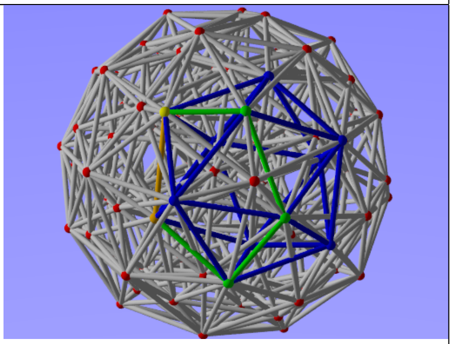
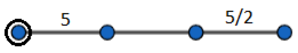
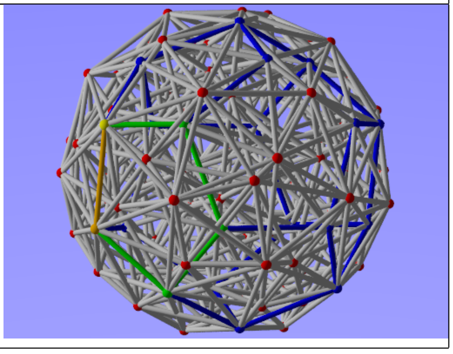
8	 or 	
	Stellated 3D polytopes	
9	 or 	
10	 or 	
11	 or 	

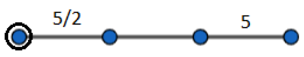
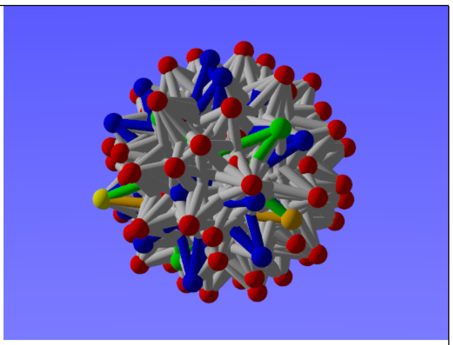
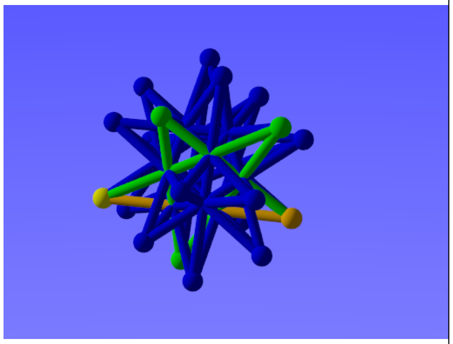
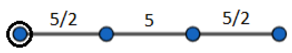
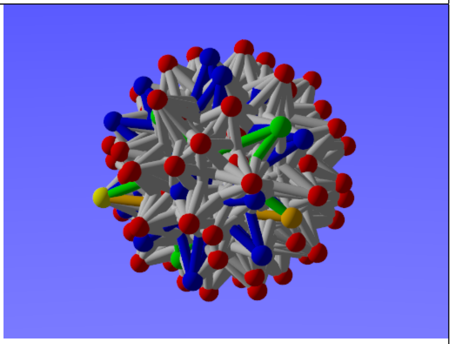
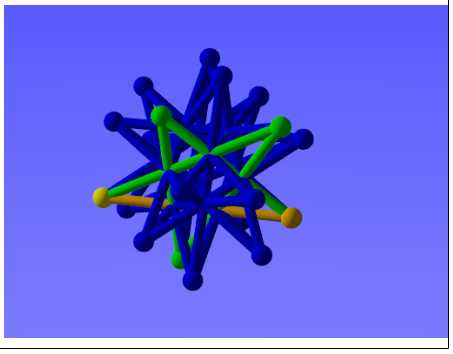
12	 or 	
13	 or 	
14	 or 	
	4D diagrams:	
15	 or 	

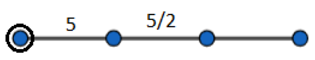
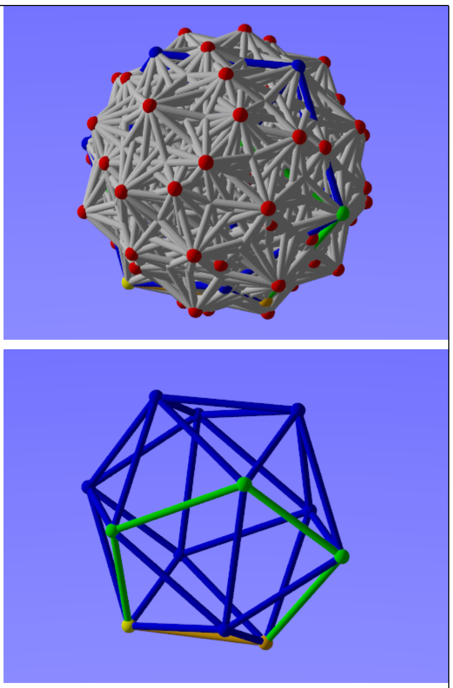
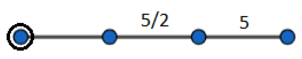
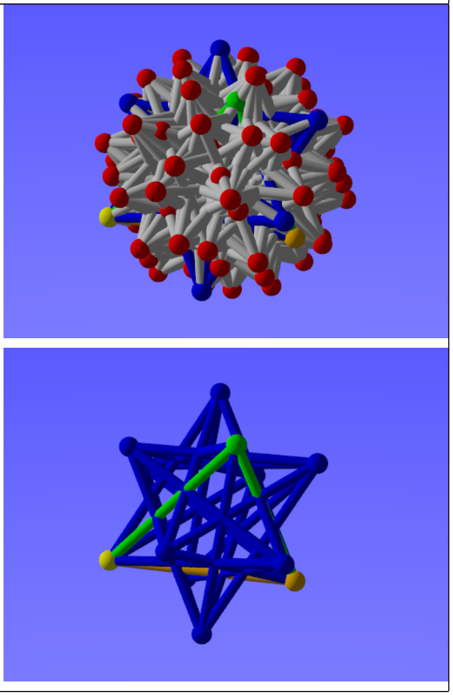
16		
17		
18		
19		

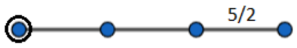
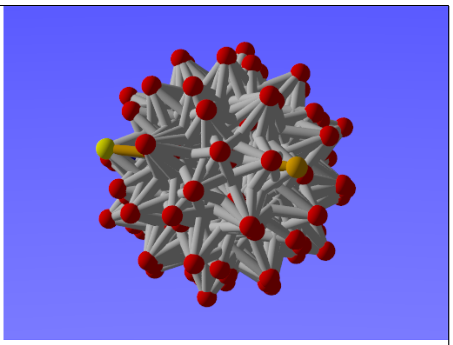
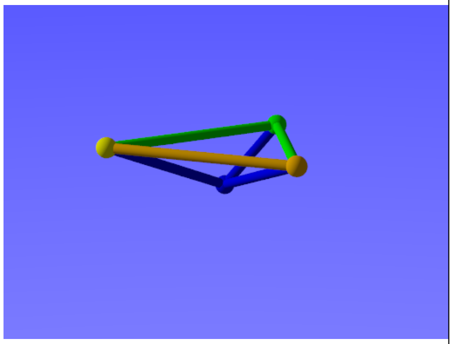
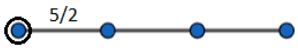
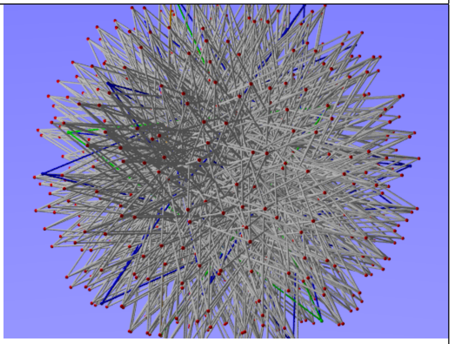
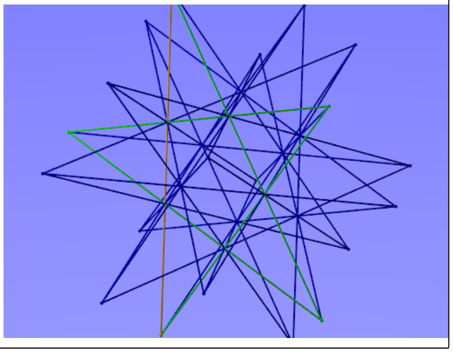
20	 or 	
21	 or 	
22	 or 	
23	 or 	

24	 or 	
25	 or 	
26	 or 	
	Stellated 4D polytopes	
27		

28		
29		
30		

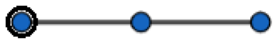
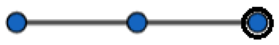
31		
32		

33		
34		

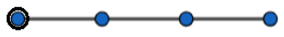
35		 
36		 

6.4.1 Relations

The 3-dimensional polytope 1 corresponding to the diagram

 or 

occurs as a cell in the 4-dimensional polytopes $15, 16, 18, 20, 24, 26$ and 35. The diagram of polytope 1 occurs in the diagrams of these polytopes. For instance it occurs in this diagram for polytope 15:

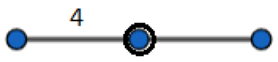


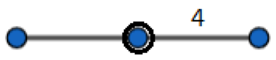
.

We see the same type of relation for the following polytopes:

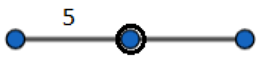
- Polytope 2 and 5 occur as cells in the polytopes 19 and 21.
- Polytope 3 occurs as cells in the polytopes $17, 22$ and 35.
- Polytope 6 occurs as cells in the polytopes 23 and 30.
- Polytope 8 occurs as cells in the polytopes $25, 27, 29$ and 33.

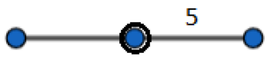
Furthermore, we can see in polytope 4 that it contains 2 different types of faces. In its diagram



or


we see on one side of the encircled node an edge marked by 4 and on the other side an unmarked edge. These are equivalent to the diagram of a polygon on respectively 4 or 3 vertices.
We see that the two types of faces are a square and a regular triangle, which are indeed representative polygons for such diagrams.
For polytope 7 this is also clear to see, here we see that the faces are a regular pentagon and a regular triangle. Which correspond to the edges next to the encircled node of its diagram being labeled 5 and unlabeled:



or


For the polytopes $10, 13, 16, 18, 19, 22, 24$ and 25 it is harder to see in the figures, however if we consider their diagrams we can expect the same type of relation.

Index

Chapter 7

Appendix

7.1 POV-Ray code

7.1.1 Code for plotting a polytope

```
#include "colors.inc"
#include "transforms.inc"

                    // change the following file names if you want to
                        use my precomputed files
#declare FLOATS       = "Floats.pov"
#declare VERTICES     = "Vectoren.pov"
#declare EDGES        = "VTXT.pov"
#declare FACES        = "ETXT.pov"
#declare CELLS        = "FTXT.pov"



// camera setup
camera {
          location <18,10,1>       // some examples: for {5,3,3} use
              <60,30,5>, for {3,3,5} use <25,15,15>, for {5/2,3,3} use
              <120,30,5>
          look_at <0,0,0>
          angle 15
  }



// the following makes sure that there is a light source and that the
     background of the pictures have a color
  light_source {
    <4, 6, 10>, White
    shadowless
  }
  sky_sphere {
    pigment {
          gradient y
          color_map{
          [0 color White]
          [1 color Blue]
          }
          scale 2
    }
  }
```

```
/* In the following piece of code we define the variables that we need to know */

#fopen Floats FLOATS read       // read and rename the file FLOATS, which is declared at the beginning of the script

#declare float    = array[9];   // initialyze the variable vector: float
#declare float[0] = 0;


#declare i = 0;
#while (defined(Floats)&i<9)    // while the read file has input we do:
   #read(Floats,float[i])           // read the i'th entry seperated by comma's from the file named Floats
   #declare i=i+1;
#end

#fclose Floats                  // stop reading the file Floats

// Set variables with the results from the float file

#if (float[0] = 0)              // if there is only one type of faces then the following are our variables

    #declare nrEdgesFace  = float[1]; // nr of edges used for each Face
    #declare nrFacesCell  = float[2]; // nr of faces used for each Cell
    #declare nrVertTotal  = float[3]; // total nr of vertices in the polytope
    #declare nrEdgesTotal = float[4]; // total nr of edges in the polytope
    #declare nrFacesTotal = float[5]; // total nr of faces in the polytope
    #declare nrCellsTotal = float[6]; // total nr of cells in the polytope

#else                           // if there are two types of faces then the following are our variables

    #declare nrEdgesFace    = float[1]; // nr of edges used for each Face that is an image of F1
    #declare nrEdgesFace_2  = float[2]; // nr of edges used for each Face that is an image of F2
    #declare nrFacesCell    = float[3]; // nr of faces used for each Cell
    #declare nrVertTotal    = float[4]; // total nr of vertices in the polytope
    #declare nrEdgesTotal   = float[5]; // total nr of edges in the polytope
    #declare nrFacesTotal   = float[6]; // total nr of faces in the polytope that are an image of F1
    #declare nrFacesTotal_2 = float[7]; // total nr of faces in the polytope that are an image of F2
    #declare nrCellsTotal   = float[8]; // total nr of cells in the polytope

#end







/* In the following piece of code we define the coordinates of all the vertices */


#fopen Coordinates VERTICES read                     // read and rename the file VERTICES, which is declared at the beginning of the script

#declare coord    = array[nrVertTotal];              // initialyze the variable vector: coord
#declare coord[0] = <0,0,0>;
#declare i = 0;
#while (defined(Coordinates) & i < nrVertTotal)     // while the file Coordinates is defined and (as double check) while i is less than the total number of vertices we do:
  #read(Coordinates,coord[i])                        // read the i'th entry seperated by comma's from the file named Coordinates
  #declare i = i+1;
#end

#fclose Coordinates                                 // stop reading the file Coordinates




```

```






/* In the following piece of code we define the vertices */

#declare Vertices      = array[nrVertTotal];   // we define the lists of vertices five times, with different sphere sizes to make sure each plotted vertex sphere shows only one color
#declare Verticesv2    = array[nrVertTotal];      // list 2 is used for the faces
#declare Verticesv3    = array[nrVertTotal];      // list 3 is used for the cells
#declare Verticesv4    = array[nrVertTotal];      // list 4 is used for coloring a single edge with its vertices
#declare Verticesv5    = array[nrVertTotal];      // list 5 is used for coloring a single vertex
#declare Vertices[0]   = sphere { <0,0,0>, .098 };
#declare Verticesv2[0] = sphere { <0,0,0>, .1 };
#declare Verticesv3[0] = sphere { <0,0,0>, .099 };
#declare Verticesv4[0] = sphere { <0,0,0>, .101 };
#declare Verticesv5[0] = sphere { <0,0,0>, .102 };
#declare i = 0;
#while (i < nrVertTotal)                                                  // while i is less than the total number of vertices do:
    #declare Vertices[i]   = sphere { coord[i], .098 pigment { Red } };      // the i'th element of the array Vertices is set to be a sphere at the i'th entry of the array with coordinates,
                                                                                                                                    //
                                                                                                                                      th
                                                                                                                                      sp
                                                                                                                                      ha
                                                                                                                                      ra
                                                                                                                                      0
                                                                                                                                      an
                                                                                                                                      is
                                                                                                                                      co
```

re

```
    #declare Verticesv2[i] = sphere { coord[i], .1 };                                // the extra lists don't get a pre set pigment so that we can set a specific color later on
    #declare Verticesv3[i] = sphere { coord[i], .099 };
    #declare Verticesv4[i] = sphere { coord[i], .101 };
    #declare Verticesv5[i] = sphere { coord[i], .102 };
    #declare i = i+1;
#end















/* In the following piece of code we define the edges */


#fopen Edgeset EDGES read                    // read and rename the file EDGES, which is declared at the beginning of the script

// Read data
#declare Edges      = array[nrEdgesTotal]; // we define the lists of edges three times, with the different sphere sizes (of the vertices) and cylinder sizes to make sure each plotted vertex and edge shows only one color
#declare Edgesv2    = array[nrEdgesTotal];     // list 2 is used for the faces
#declare Edgesv3    = array[nrEdgesTotal];     // list 3 is used for the cells
#declare Edgesv4    = array[nrEdgesTotal];     // list 4 is used for coloring a single edge with its vertices
#declare Edges[0]   = union{};
#declare Edgesv2[0] = union{};
#declare Edgesv3[0] = union{};
#declare Edgesv4[0] = union{};

#declare i = 0;

#while (defined(Edgeset) & i < nrEdgesTotal)   // while the file
```

```
     Edgeset is defined and (as extra check) while i is smaller than
     the total number of edges do:
    #read(Edgeset,vert1,vert2)                    // read the next pair
       of entries from the file Edgeset

    #declare Edges[i]   = union{ object{Vertices[vert1 1]}         //
       define the i'th entry of the array Edges to be the pair of
       entries from the array Vertices that have just been read from
       Edgeset
                                 object{Vertices[vert2 1]}         //
                                    we take the entry vert2 1 because
                                    Mathematica's numbers for the
                                    entries start from 1,... but POV
                                    Ray uses 0,1...
                                 cylinder{ coord[vert1 1],coord[vert2
                                    1], .0498 pigment{ Grey }  }
                                    // and the i'th entry also consist
                                     of a cylinder in between these
                                    vertex coordinates, the cylinder
                               };

                                  // has radius 0.0498 and is colored
                                  grey
    #declare Edgesv2[i] = union{ object{Verticesv2[vert1 1]}
                                 object{Verticesv2[vert2 1]}
                                 cylinder{ coord[vert1 1],coord[vert2
                                    1], .05 }   // the extra lists
                                    don't get a pre set pigment so
                                    that we can set a specific color
                                    for a face or cell
                               };
    #declare Edgesv3[i] = union{ object{Verticesv3[vert1 1]}
                                 object{Verticesv3[vert2 1]}
                                 cylinder{ coord[vert1 1],coord[vert2
                                    1], .0499 }
                               };
    #declare Edgesv4[i] = union{ object{Verticesv4[vert1 1]}
                                 object{Verticesv4[vert2 1]}
                                 cylinder{ coord[vert1 1],coord[vert2
                                    1], .0501 }
                               };
    #declare i = i+1;
#end


#fclose Edgeset                         // stop reading the file Edgeset



```

```
/* In the following piece of code we define the faces */


#fopen Faceset FACES read           // read and rename the file FACES,
     which is declared at the beginning of the script

                          /* Below we define a list of edges in the
                              right order to construct all faces with
                              them */
#if (float[0] = 0)
    #declare faceend = nrEdgesFace*nrFacesTotal;
#else
    #declare faceend = (nrEdgesFace*nrFacesTotal)+(nrEdgesFace_2*
        nrFacesTotal_2);     // the list of edges has a different
        length if we have two types of faces
#end

#declare faceedges    = array[faceend];     // we define the lists of
    faceedges two times to make sure each plotted vertex and edge
    shows only one color
#declare faceedges2    = array[faceend];        // list 2 is used for
    the cells
#declare faceedges[0]  = union{};
#declare faceedges2[0] = union{};
#declare i = 0;

#while (defined(Faceset) & i < faceend)        // while the file
    Faceset is defined and (as axtra check) while i is smaller than
    the length of the array faceedges do:
   #read(Faceset,edge)                              // read the next
       entry from the file Faceset, each entry is an edge number
   #declare faceedges[i]= Edgesv2[edge 1];          // each entry of
       the array faceedges contains an element of edgesv2, again using
        edge 1 because of the difference in array numbering between
       Mathematica and POV Ray
   #declare faceedges2[i]= Edgesv3[edge 1];
   #declare i = i+1;
#end

#fclose Faceset                              // stop reading the file
```

```
    Faceset

                             /* Below we construct the faces from the
                                edges that represent them */

#if (float[0] = 0)                                      // if there is
    only one type of faces
    #declare Faces      = array[nrFacesTotal];          // This is the
        list of faces, we again define the list twice for later
        purposes
    #declare Faces2     = array[nrFacesTotal];              // list 2 is
        again used for the cells
    #declare Faces[0]   = union{};
    #declare Faces2[0]  = union{};
    #declare j=0;
    #while (j<nrFacesTotal)
        // while j is less than the total number of faces do:
        #declare Faces[j] = union{ #declare k=j*nrEdgesFace;
                            // the j'th entry of Faces is the union of the
            edges in the entries k,...,k+(number of edges per face)
                                  #while (k<(j+1)*nrEdgesFace)
                                      object{faceedges[k]}
                                                 // each object
                                         in faceedges is an edge (
                                         two vertices (spheres) and
                                          a cylinder in between)
                                      #declare k=k+1;
                                  #end
                                 };
        #declare Faces2[j] = union{ #declare k=j*nrEdgesFace;
                                   #while (k<(j+1)*nrEdgesFace)
                                      object{faceedges2[k]}
                                      #declare k=k+1;
                                   #end
                                 };
        #declare j=j+1;
    #end

#else                                                     // if there are
     two types of faces
    #declare Faces      = array[nrFacesTotal+nrFacesTotal_2];
        // This is the list of faces, we again define the list twice
        for later purposes
    #declare Faces2     = array[nrFacesTotal+nrFacesTotal_2];
                   // list 2 is again used for the cells
    #declare Faces[0]   = union{};
    #declare Faces2[0]  = union{};
    #declare j=0;
    #while (j<nrFacesTotal)
```

```
                                                              // while j is less
    than the total number of faces do:
        #declare Faces[j] = union{ #declare k=j*nrEdgesFace;
                            // the j'th entry of Faces is the union of the
            edges in the entries k,...,k+(number of edges per face)
                                      #while (k<(j+1)*nrEdgesFace)
                                          object{faceedges[k]}
                                                      // each object
                                               in faceedges is an edge (
                                              two vertices (spheres) and
                                               a cylinder in between)
                                          #declare k=k+1;
                                      #end
                                    };
        #declare Faces2[j] = union{ #declare k=j*nrEdgesFace;
                                      #while (k<(j+1)*nrEdgesFace)
                                          object{faceedges2[k]}
                                          #declare k=k+1;
                                      #end
                                    };
        #declare j=j+1;
    #end

    #declare n = nrFacesTotal*nrEdgesFace;                          //
        since there are two types of faces, they might contain a
        different number of edges per face
    #declare l = 0;                                                 // we
        start to count again from the total number of faces of type 1
        multiplied by how many edges are in this type of faces
    #while (l<nrFacesTotal_2)
                                                              // while l is less
    than the total number of faces do:
        #declare Faces[nrFacesTotal+l] = union{ #declare k=n+l*
            nrEdgesFace_2;        // the (total number of faces of
            type 1 + l)'th entry of Faces is the union of the edges in
            the entries k,...,k+(number of edges per face)
                                                   #while (k<n+(l+1)*
                                                      nrEdgesFace_2)
                                                       object{faceedges[
                                                          k]}

                                                          // each object
                                                           in faceedges
                                                          is an edge (
                                                          two vertices (
                                                          spheres) and a
                                                           cylinder in
                                                          between)
                                                      #declare k=k+1;
```

```
                                                 #end
                                                };
        #declare Faces2[nrFacesTotal+l] = union{ #declare k=n+l*nrEdgesFace_2;
                                                  #while (k<n+(l+1)*nrEdgesFace_2)
                                                      object{faceedges2[k]}
                                                      #declare k=k+1;
                                                  #end
                                                };
        #declare l=l+1;
    #end
#end











/* In the following piece of code we define the cells */


#fopen Cellset CELLS read          // read and rename the file CELLS, which is declared at the beginning of the script

                         /* Below we define a list of faces in the right order for all cells */

#declare cellfaces    = array[nrFacesCell*nrCellsTotal];  // initialize the list cellfaces, in this list we will store all face numbers in the right order to define the cells
#declare cellfaces[0] = union{};                            // there is no need anymore to define the list multiple times

#declare i = 0;

#while (defined(Cellset) & i < nrFacesCell*nrCellsTotal)     // while the file Cellset is defined and (as extra check) while i is less than (the total number of cells)*(number of faces per cell) do:
   #read(Cellset,face)                                        // read the next entry from the file Cellset, each entry is a face number
```

```
    #declare cellfaces[i]= Faces2[face 1];                        // the i'th entry of cellfaces consists of a face, again using face 1 as index to make up for the difference between Mathematica and POV Ray
    #declare i = i+1;
#end

#fclose Cellset                        // stop reading the file Cellset

                              /* Below we construct the cells from the faces that represent them */

#declare Cells    = array[nrCellsTotal]; // Create and initialize the array Cells
#declare Cells[0] = union{};

#declare j = 0;

#while ( j < nrCellsTotal)    //while j is less than the total number of cells do:
    #declare Cells[j] = union{  #declare k=j*nrFacesCell;                    // the j'th entry the array Cells consists of the cellfaces k,...,(j+1)*(number of faces per cell) 1
                                   #while (k<(j+1)*nrFacesCell)
                                          object{cellfaces[k]}                         // each entry of cellfaces is a face (multiple edges)
                                          #declare k=k+1;
                                   #end
                                  };
    #declare j = j+1;
#end
```

```








    /* Now that everything is defined, below we can define everything that we want to be shown in the figures and what colors should be used */




  /* first make a decision in the following: */
   #declare show_polytope = on; // set to off if you do not want to show the basic structure (vertices and edges) of the polytope
   #declare show_initial_flag = off; // set to off if you do not want to show an initial maximal flag of the polytope






// Everything that you want to have shown, put it inside of the union {...}
union
{

                // This makes sure that all vertices and edges are shown
                   #if (show_polytope)                        // if show_polytope is set to 'on', then show all vertices and edges
                     #declare k = 0;
                     #while (k < nrVertTotal)
                        object{Vertices[k] }
                        #declare k = k+1;
                     #end
                     #declare k = 0;
                     #while (k < nrEdgesTotal)
                        object{Edges[k]  }
                        #declare k = k+1;
                     #end
                   #end

                // This makes sure that a maximal flag is shown
                   #if (show_initial_flag)                  // if
                show_initial_flag is set to 'on', then give the
                elements of the initial flag different colors
              object{ Verticesv5[0] pigment{ Yellow } }
              object{ Edgesv4[0]    pigment{ Orange } }
              object{ Faces[0]      pigment{ Green  } }
              object{ Cells[0]      pigment{ Blue   } }
          #end



      // With the following examples you can choose which
         faces or cells you want to give a different color
           // object{Verticesv5[nr]  pigment{ Color } }
           // object{Edgev4[nr]      pigment{ Color } }
           // object{Faces[nr]       pigment{ Color } }
           // object{Cells[nr]       pigment{ Color } }

           // copy any of the 4 examples above and paste them
                below this line






      // with the following we can rotate our figure such
         that we get a nice view

           rotate <0+360*(clock),0,0>

}
```

7.1.2 Code for spinning polytopes

```
; POV Ray animation ini file
Antialias=Off
Antialias_Threshold=0.1
Antialias_Depth=2

Input_File_Name="4DFigures.pov"

Initial_Frame=1
Final_Frame=100
Initial_Clock=0
Final_Clock=1
Cyclic_Animation=on
```

7.2 Mathematica code

See from the next page and onwards

```
(* First make a screen where the user can give a desired Schlafli Symbol
 together with a reflection hyperplane they want to start reflecting in *)
Style["Give the values of p,q,r for a Schlafli symbol of the form {p,q,r}",
 16, Italic, Bold, Blue, FontFamily → "Courier", LineSpacing → {1, 0}]
Style["p" InputField[Dynamic[pp]], 14, Italic,
 FontFamily → "Courier", LineSpacing → {1, 0}]
Style["q" InputField[Dynamic[qq]], 14, Italic,
 FontFamily → "Courier", LineSpacing → {1, 0}]
Style["r" InputField[Dynamic[rr]], 14, Italic,
 FontFamily → "Courier", LineSpacing → {1, 0}]
Style["Choose a node of the diagram to start with the corresponding reflection",
 16, Italic, Bold, Blue, FontFamily → "Courier", LineSpacing → {1, 0}]
Style["Type 1,2,3 or 4: " InputField[Dynamic[k]], 14,
 Italic, FontFamily → "Courier", LineSpacing → {1, 0}]
Style["Give the directory where you want to write all files to", 16,
 Italic, Bold, Blue, FontFamily → "Courier", LineSpacing → {1, 0}]
Style["Give the directory: " InputField[Dynamic[directory], FieldSize → 50],
 14, Italic, FontFamily → "Courier", LineSpacing → {1, 0}]
```

Out[ ]= ***Give the values of p,q,r***
for a Schlafli symbol of the form {p,q,r}

Out[ ]= *p* | *pp* |

Out[ ]= *q* | *qq* |

Out[ ]= *r* | *rr* |

Out[ ]= ***Choose a node of the diagram***
to start with the corresponding reflection

Out[ ]= *Type 1,2,3 or 4:* | *k* |

Out[ ]= ***Give the directory where you want to write all files to***

Out[ ]= *Give the directory:*
| *directory*

```
Style["Are these the values you wanted?", 16, Italic,
 Bold, Blue, FontFamily → "Courier", LineSpacing → {1, 0}]
Schl = {pp, qq, rr}     (* This section is just to double
 check the values that have been given as input *)
k
```

Out[ ]= ***Are this the values you wanted?***

Out[ ]= {3, 3, 3}

Out[ ]= 1

```
(* With the choosen input we can determine the
 normal vectors a_i of the reflection hyperplanes r_i *)
a = Cos[Pi / Schl[[1]]];   (* define the angles between
 the hyperplanes with the given Schläfli symbol *)
b = Cos[Pi / Schl[[2]]];
c = Cos[Pi / Schl[[3]]];
a1 = {1, 0, 0, 0};          (* We choose the first vector *)
a2 = {a, n1, 0, 0};
(* For the other vectors we make sure that the angle relations are met *)
s3 = FindInstance[Norm[a2] == 1, {n1}, Reals];      (* and then determine
 the value for the variable n_ such that the vectors have length 1 *)
a2 = Replace[a2 /. s3, {x_List} :→ x, {0, -3}];
a3 = {0, b / a2[[2]], n2, 0};
s1 = FindInstance[Norm[a3] == 1, {n2}, Reals];
a3 = Replace[a3 /. s1, {x_List} :→ x, {0, -3}];
a4 = {0, 0, c / a3[[3]], n3};
s2 = FindInstance[Norm[a4] == 1, {n3}, Reals];
a4 = Replace[a4 /. s2, {x_List} :→ x, {0, -3}];
Style["The following are the normal vectors a_i that we will use in our calculation",
 16, Italic, Bold, Blue, FontFamily → "Courier", LineSpacing → {1, 0}]
a1
a2 = Simplify[a2]        (* this is to show the computed vectors *)
a3 = Simplify[a3]
a4 = Simplify[a4]
Style[
 "The following must be True in order for us to know if all restrictions are met",
 16, Italic, Bold, Blue, FontFamily → "Courier", LineSpacing → {1, 0}]
Simplify[a1.a2] == a && Simplify[a1.a3] == 0 && Simplify[a1.a4] == 0 &&
 Simplify[a2.a3] == b && Simplify[a2.a4] == 0 && Simplify[a3.a4] == c &&
 Simplify[Norm[a1]] == 1 && Simplify[Norm[a2]] == 1 && Simplify[Norm[a3]] == 1 &&
 Simplify[Norm[a4]] == 1          (* this is a check for all relations that should
   hold: the angles between each pair of normal vectors and their unit length*)
```

Out[∘]= ***The following are the normal vectors a_i that we will use in our calculation***

Out[∘]= $\{1, 0, 0, 0\}$

Out[∘]= $\left\{\frac{1}{2}, \frac{\sqrt{3}}{2}, 0, 0\right\}$

Out[∘]= $\left\{0, \frac{1}{\sqrt{3}}, \sqrt{\frac{2}{3}}, 0\right\}$

Out[∘]= $\left\{0, 0, \frac{\sqrt{\frac{3}{2}}}{2}, \frac{\sqrt{\frac{5}{2}}}{2}\right\}$

Out[∘]= ***The following must be True in order for us to know if all restrictions are met***

Out[∘]= True

```
A = Join[{a1}, {a2}, {a3}, {a4}];        (* For the ease of computations
 we will use the normal vectors as rowvectors of a matrix A *)
p = {p1, p2, p3, p4};
s2 = FindInstance[If[k ≠ 1, p.A[[1]] == 0, p ≠ {0, 0, 0, 0}] &&
     If[k ≠ 2, p.A[[2]] == 0, p ≠ {0, 0, 0, 0}] && If[k ≠ 3, p.A[[3]] == 0, p ≠ {0, 0, 0, 0}] &&
     If[k ≠ 4, p.A[[4]] == 0, p ≠ {0, 0, 0, 0}], {p1, p2, p3, p4}];
(* This finds a vector p contained in the three hyperplanes other
 than the desired starting reflection hyperplane*)
ps = p /. s2;              (* This applies the result
 found for p to define v_1 as our first vertex *)
Style["The following vector is our first vertex of the polytope, v_1",
 16, Italic, Bold, Blue, FontFamily → "Courier", LineSpacing → {1, 0}]
v1 = FullSimplify[ps[[1]]]       (* This shows the vertex v_1 to the user *)
```

Out[ ]= ***The following vector is our first vertex of the polytope, v_1***

Out[ ]= $\left\{1, -\frac{1}{\sqrt{3}}, \frac{1}{\sqrt{6}}, -\frac{1}{\sqrt{10}}\right\}$

```
(* Let us define the reflection transformation functions rt_i
 that represent reflecting in the hyperplanes perpendicular to a_i *)
rt1 = ReflectionTransform[A[[1]]];
rt2 = ReflectionTransform[A[[2]]];
rt3 = ReflectionTransform[A[[3]]];
rt4 = ReflectionTransform[A[[4]]];
(* For the ease of coding we define the
 reflection transformations as elements of a list RT *)
RT = {rt1, rt2, rt3, rt4};
(* We use the normal vector of the desired starting reflection
 hyperplane to reflect v1 and in this way we find an edge *)
Style["The following vector is our second vertex of the polytope, v_2",
 16, Italic, Bold, Blue, FontFamily → "Courier", LineSpacing → {1, 0}]
v2 = RT[[k]][v1]        (* This computes v_2 and shows the vertex v_2 to the user *)
```

Out[ ]= ***The following vector is our second vertex of the polytope, v_2***

Out[ ]= $\left\{-1, -\frac{1}{\sqrt{3}}, \frac{1}{\sqrt{6}}, -\frac{1}{\sqrt{10}}\right\}$

```
ClearAll[Vertices]
(* Define the list Vertices to keep track
 of all the vertices that we find by reflecting v_1 *)
Vertices = {v1, v2};
vi = 1;      (* initialize a counter *)

(* Define the function that we will use to check
 whether a vertex is already a member of our list *)
notmember[list_, x_] := For[i = 1, i ≤ Length[list], i++,
  If[Norm[list[[i]] - x] < 0.1, output = False; Break[], output = True;]]
(* This function checks if a vector is not an element of a vector list,
if it is not a member this function returns True *)
(* This function is later also used for finding an initial face and cell *)

(* In the while loop we reflect the vertex v_1 in
 4 reflection axes until we do not find any new vertices *)
While[vi <= Length[Vertices],
 nv = N[Vertices];
 (* This is a numerical representation of all vertices that we have found so far *)

 mapv1 = FullSimplify[RT[[1]][Vertices[[vi]]]];    (* This computes an image
  after reflecting a vertex of the polytope in the first hyperplane *)
 notmember[nv, N[mapv1]];                          (* This checks whether this
  image is already contained in our list of vertices using the function notmember *)
 If[output,
  Vertices = Join[Vertices, {mapv1}]                (* If the function notmember
   returns the output True then the computed image is added to the vertices list *)
 ];

 mapv2 = FullSimplify[RT[[2]][Vertices[[vi]]]];     (* From here we do
  the same as before but for the other three hyperplane reflections *)
 notmember[nv, N[mapv2]];
 If[output,
  Vertices = Join[Vertices, {mapv2}]
 ];
 mapv3 = FullSimplify[RT[[3]][Vertices[[vi]]]];
 notmember[nv, N[mapv3]];
 If[output,
  Vertices = Join[Vertices, {mapv3}]
 ];
 mapv4 = FullSimplify[RT[[4]][Vertices[[vi]]]];
 notmember[nv, N[mapv4]];
 If[output,
  Vertices = Join[Vertices, {mapv4}]
 ];
 vi++   (* increase the counter *)
]
Style["The following is the number of vertices contained in our polytope",
 16, Italic, Bold, Blue, FontFamily → "Courier", LineSpacing → {1, 0}]
Length[Vertices] (* This counts the number of vertices in our polytope *)
Vertices;
```

```
Out[•]= The following is the number
          of vertices contained in our polytope
```

```
Out[∘]= 5

ClearAll[Edges]
Style["The following is our first edge of the polytope, e_1 connecting v_1 and v_2",
 16, Italic, Bold, Blue, FontFamily → "Courier", LineSpacing → {1, 0}]
E1 = SortBy[FullSimplify[{v1, v2}], Less]
(* The first edge we find is an edge connecting the vertices v1 and v2,
we show it to the user *)
(* Define the list Edges to keep track of the edges that
 we find by reflecting e_1 that are contained in our polytope *)
Edges = {E1};
(* Define the function that will be used to check whether a list of elements already
 contains the element in question (used for either edges, faces or cells *)
notmember2[list_, list2_] := For[i = 1, i ≤ Length[list], i++,
                                count = 0;
                                For[j = 1, j ≤ Length[list[[1]]], j++,
                                     If[Norm[list[[i, j]] - list2[[j]]] < 0.1,
                                              count = count + 1;
                                      ];
                                     If[count == Length[list[[1]]], output = False;
    Break[], output = True]
                                 ];
                                If[output == False, Break[]]
  (* If the output is True then the element is not a member of the list *)
                              ]

(* In the while loop we reflect the edge e_1
 in 2 reflection axes until we do not find a new edge *)
ei = 1;   (* Initialize a counter *)
While[ei ≤ Length[Edges],
 ne = N[Edges];
 (* This is a numerical representation of all edges that we have found so far *)

 mape1 = SortBy[RootReduce[FullSimplify[Map[RT[[1]], Edges[[ei]]]]], Less];
 (* This computes an image after reflecting
  an edge of the polytope in the first hyperplane *)
 notmember2[ne, N[mape1]];        (* This checks whether this image is
  already contained in our list of vertices using the function notmember2 *)
 If[output,
  Edges = Join[Edges, {mape1}]    (* If the function notmember2 returns
   the output True then the computed image is added to the edges list *)
 ];

 mape2 = SortBy[RootReduce[FullSimplify[Map[RT[[2]], Edges[[ei]]]]], Less];
 (* From here we do the same as before but
  for the other three hyperplane reflections *)
 notmember2[ne, N[mape2]];
 If[output,
  Edges = Join[Edges, {mape2}]
 ];
 mape3 = SortBy[RootReduce[FullSimplify[Map[RT[[3]], Edges[[ei]]]]], Less];
 notmember2[ne, N[mape3]];
 If[output,
  Edges = Join[Edges, {mape3}]
 ];
 mape4 = SortBy[RootReduce[FullSimplify[Map[RT[[4]], Edges[[ei]]]]], Less];
 notmember2[ne, N[mape4]];
 If[output,
  Edges = Join[Edges, {mape4}]
 ];
 ei++  (* increase the counter *)
]
Style["The following is the number of edges contained in our polytope",
 16, Italic, Bold, Blue, FontFamily → "Courier", LineSpacing → {1, 0}]
Length[Edges] (* This counts the number of edges
 in our first face of the polytope *)
Edges;
```

Out[∘]= *The following is our first edge*
of the polytope, e_1 connecting v_1 and v_2

Out[∘]= $\left\{\left\{-1, -\frac{1}{\sqrt{3}}, \frac{1}{\sqrt{6}}, -\frac{1}{\sqrt{10}}\right\}, \left\{1, -\frac{1}{\sqrt{3}}, \frac{1}{\sqrt{6}}, -\frac{1}{\sqrt{10}}\right\}\right\}$

Out[∘]= *The following is the number of edges contained in our polytope*

Out[∘]= 10

```
ClearAll[F1, F2, fi, Faces, Faces1, Faces2]
(* the first Face F_1 we find is found by
 using two reflections and applying these to v1 *)
n = 1;
F1 = {v1, v2};
F2 = {v1, v2}; (* In the case that k=
 2 or 3 we might find 2 different types of faces and thus we will initialize both *)

(* We need to check the value of k (on which node the user wanted to encircle) *)
If[k == 1,
 While[n ≤ Length[F1],     (* In the while loop we reflect our first vertices v1 and
   v2 with a selection of the reflection functions to determine an initial face *)
  mapf1 = FullSimplify[RT[[1]][F1[[n]]]];
  notmember[N[F1], N[mapf1]];
  (* Recall that we use the predefined function notmember,
  if output is true than the element is not a member of the list*)
  If[output,
   F1 = Join[F1, {mapf1}]
  ];
  mapf2 = FullSimplify[RT[[2]][F1[[n]]]];
  notmember[N[F1], N[mapf2]];
  If[output,
   F1 = Join[F1, {mapf2}]
  ];
  n++
 ],
 If[k == 2,
  While[n ≤ Length[F2],
   mapf1 = FullSimplify[RT[[1]][F2[[n]]]];
   (* Note that for k=2 we define F2 first to make sure that F1 is the
     type of faces that is associated with the cells of the polytope*)
   notmember[N[F2], N[mapf1]];
   If[output,
   F2 = Join[F2, {mapf1}]
  ];
  mapf2 = FullSimplify[RT[[2]][F2[[n]]]];
  notmember[N[F2], N[mapf2]];
  If[output,
   F2 = Join[F2, {mapf2}]
  ];
  n++
 ];
 n = 1;
 While[n ≤ Length[F1],
  mapf1 = FullSimplify[RT[[2]][F1[[n]]]];
  notmember[N[F1], N[mapf1]];
  If[output,
   F1 = Join[F1, {mapf1}]
  ];
  mapf2 = FullSimplify[RT[[3]][F1[[n]]]];
  notmember[N[F1], N[mapf2]];
  If[output,
   F1 = Join[F1, {mapf2}]
  ];
  n++
 ],
 If[k == 3,
  (* Note that for k=3 we define F1 first to make sure that F1 is the
    type of faces that is associated with the cells of the polytope*)
  While[n ≤ Length[F1],
   mapf1 = FullSimplify[RT[[2]][F1[[n]]]];
   notmember[N[F1], N[mapf1]];
   If[output,
    F1 = Join[F1, {mapf1}]
   ];
   mapf2 = FullSimplify[RT[[3]][F1[[n]]]];
   notmember[N[F1], N[mapf2]];
   If[output,
    F1 = Join[F1, {mapf2}]
   ];
   n++
  ];
  n = 1;
  While[n ≤ Length[F2],
   mapf1 = FullSimplify[RT[[3]][F2[[n]]]];
   notmember[N[F2], N[mapf1]];
   If[output,
    F2 = Join[F2, {mapf1}]
   ];
   mapf2 = FullSimplify[RT[[4]][F2[[n]]]];
   notmember[N[F2], N[mapf2]];
   If[output,
    F2 = Join[F2, {mapf2}]
   ];
   n++
  ],
  If[k == 4,
   While[n ≤ Length[F1],
        mapf1 = FullSimplify[RT[[3]][F1[[n]]]];
        notmember[N[F1], N[mapf1]];
        If[output,
         F1 = Join[F1, {mapf1}]
        ];
        mapf2 = FullSimplify[RT[[4]][F1[[n]]]];
        notmember[N[F1], N[mapf2]];
        If[output,
         F1 = Join[F1, {mapf2}]
        ];
        n++
       ]]]]]
If[k == 1 || k == 4, Style[
  "The following is the number of vertices/edges contained in our first face, F_1",
  16, Italic, Bold, Blue, FontFamily → "Courier", LineSpacing → {1, 0}],
 Style["The following are the numbers of vertices/edges
    contained in the first two different faces, F_1 and F_2 ",
  16, Italic, Bold, Blue, FontFamily → "Courier", LineSpacing → {1, 0}]]
If[k == 1 || k == 4, facelength = Length[F1], facelength = {Length[F1], Length[F2]}]
(* Define the number of edges in the initial face(s) and show it to the user *)
F1 = SortBy[F1, Less];
F2 = SortBy[F2, Less];

(* Reflecting the first face so that we can find all faces of the polytope *)
fi = 1; (* initialize a counter *)
If[k == 1 || k == 4,      (* check the k value *)
 Faces = {F1};

 While[fi ≤ Length[Faces], (* in this while loop we reflect the
   initial face with all reflection functions to find all the images *)
  nf = N[Faces];
  mapf1 = SortBy[RootReduce[FullSimplify[Map[RT[[1]], Faces[[fi]]]]], Less];
  notmember2[nf, N[mapf1]];
  (* Recall that we use the predefined function notmember2,
  if output is true than the element is not a member of the list*)
  If[output,
   Faces = Join[Faces, {mapf1}]
  ];
  mapf2 = SortBy[RootReduce[FullSimplify[Map[RT[[2]], Faces[[fi]]]]], Less];
  notmember2[nf, N[mapf2]];
  If[output,
   Faces = Join[Faces, {mapf2}]
  ];
  mapf3 = SortBy[RootReduce[FullSimplify[Map[RT[[3]], Faces[[fi]]]]], Less];
  notmember2[nf, N[mapf3]];
  If[output,
   Faces = Join[Faces, {mapf3}]
  ];
  mapf4 = SortBy[RootReduce[FullSimplify[Map[RT[[4]], Faces[[fi]]]]], Less];
  notmember2[nf, N[mapf4]];
  If[output,
   Faces = Join[Faces, {mapf4}]
  ];
  fi++ (* increase the counter *)
 ],
If[k == 2 || k == 3,
 Faces1 = {F1};
 While[fi ≤ Length[Faces1],
  (* in this while loop we reflect the initial face F1 with all
   reflection functions to find all the images of the type of faces F1 *)
  nf1 = N[Faces1];
  mapf1 = SortBy[RootReduce[FullSimplify[Map[RT[[1]], Faces1[[fi]]]]], Less];
  notmember2[nf1, N[mapf1]];
  If[output,
   Faces1 = Join[Faces1, {mapf1}]
  ];
  mapf2 = SortBy[RootReduce[FullSimplify[Map[RT[[2]], Faces1[[fi]]]]], Less];
  notmember2[nf1, N[mapf2]];
  If[output,
   Faces1 = Join[Faces1, {mapf2}]
  ];
  mapf3 = SortBy[RootReduce[FullSimplify[Map[RT[[3]], Faces1[[fi]]]]], Less];
  notmember2[nf1, N[mapf3]];
  If[output,
   Faces1 = Join[Faces1, {mapf3}]
  ];
  mapf4 = SortBy[RootReduce[FullSimplify[Map[RT[[4]], Faces1[[fi]]]]], Less];
  notmember2[nf1, N[mapf4]];
  If[output,
   Faces1 = Join[Faces1, {mapf4}]
  ];
  fi++   (* increase the counter *)
 ];
 fi = 1;  (* restart the counter *)
 Faces2 = {F2};
 While[fi ≤ Length[Faces2],
  (* in this while loop we reflect the initial face F2 with all
   reflection functions to find all the images of the type of faces F2 *)
  nf2 = N[Faces2];
  mapf1 = SortBy[RootReduce[FullSimplify[Map[RT[[1]], Faces2[[fi]]]]], Less];
  notmember2[nf2, N[mapf1]];
  If[output,
   Faces2 = Join[Faces2, {mapf1}]
  ];
  mapf2 = SortBy[RootReduce[FullSimplify[Map[RT[[2]], Faces2[[fi]]]]], Less];
  notmember2[nf2, N[mapf2]];
  If[output,
   Faces2 = Join[Faces2, {mapf2}]
  ];
  mapf3 = SortBy[RootReduce[FullSimplify[Map[RT[[3]], Faces2[[fi]]]]], Less];
  notmember2[nf2, N[mapf3]];
  If[output,
   Faces2 = Join[Faces2, {mapf3}]
  ];
  mapf4 = SortBy[RootReduce[FullSimplify[Map[RT[[4]], Faces2[[fi]]]]], Less];
  notmember2[nf2, N[mapf4]];
  If[output,
   Faces2 = Join[Faces2, {mapf4}]
  ];
  fi++  (* increase the counter *)
  ]
 ]]

If[k == 2 || k == 3, Faces = Join[Faces1, Faces2]];
(* Define the set of faces as the combined
  set of images of F1 and F2 if k=2 or k=3 *)
Faces;
If[k == 2 || k == 3,
 Style["The following are the number of images of F_1 and the number of
    images of F_2 contained in our polytope", 16,
  Italic, Bold, Blue, FontFamily → "Courier", LineSpacing → {1, 0}]]
If[k == 2 || k == 3, {Length[Faces1], Length[Faces2]}]
(* if k=2 or k=3 it might be nice to know
   how many of each type of faces there are in the polytope *)
Faces1;
Faces2;
Style["The following is the number of faces contained in our polytope",
 16, Italic, Bold, Blue, FontFamily → "Courier", LineSpacing → {1, 0}]
Length[Faces] (* This counts the number of edges in our first cell of the polytope *)
```

Out[∘]= ***The following is the number of vertices/edges contained in our first face, F_1***

Out[∘]= 3

Out[∘]= ***The following is the number of faces contained in our polytope***

Out[∘]= 10

```
ClearAll[C1, ci, Cels]
(* The initial Cell C1 is found by using three
 reflections and applying these our initial vertex v1 *)
C1 = {v1};
m = 1;
If[k == 1 || k == 2,
 While[m ≤ Length[C1],
  mapc1 = FullSimplify[RT[[k]][C1[[m]]]];
  notmember[N[C1], N[mapc1]];       (* We again use our predefined function
   notmember to check whether the image of a vertex was already in our list *)
  If[output,
   C1 = Join[C1, {mapc1}]
  ];
  mapc2 = FullSimplify[RT[[k + 1]][C1[[m]]]];
  notmember[N[C1], N[mapc2]];
  If[output,
   C1 = Join[C1, {mapc2}]
  ];
  mapc3 = FullSimplify[RT[[k + 2]][C1[[m]]]];
  notmember[N[C1], N[mapc3]];
  If[output,
   C1 = Join[C1, {mapc3}]
  ];
  m++
 ],
 If[k == 3 || k == 4,
  While[m ≤ Length[C1],
    mapc1 = FullSimplify[RT[[k]][C1[[m]]]];
    notmember[N[C1], N[mapc1]];
    If[output,
     C1 = Join[C1, {mapc1}]
    ];
    mapc2 = FullSimplify[RT[[k - 1]][C1[[m]]]];
    notmember[N[C1], N[mapc2]];
    If[output,
     C1 = Join[C1, {mapc2}]
    ];
    mapc3 = FullSimplify[RT[[k - 2]][C1[[m]]]];
    notmember[N[C1], N[mapc3]];
    If[output,
     C1 = Join[C1, {mapc3}]
    ];
    m++
   ]
  ]]
Style["The following is the number of vertices contained in our first cell C_1",
 16, Italic, Bold, Blue, FontFamily → "Courier", LineSpacing → {1, 0}]
Length[C1] (* This counts the number of vertices in C1 *)

(* Now that we have the first Cell we
 can determine all other cells of the polytope*)
Cels = {SortBy[RootReduce[FullSimplify[C1]], Less]};
ci = 1;
While[ci ≤ Length[Cels],
 (* In this while loop we reflect the initial cell C1 in all hyperplanes,
 then reflect all its images, etcetera to find all cells of the polytope *)
 nc = N[Cels];
 mapc1 = SortBy[RootReduce[FullSimplify[RT[[1]][Cels[[ci]]]]], Less];
 (* reflect a cell in the first hyperplane *)
 notmember2[nc, N[mapc1]];
 (* Check whether we already had this image in our list of cells *)
 If[output,
  Cels = Join[Cels, {mapc1}]
  (* If this was not the case then add the new cell to the list*)
 ];
 mapc2 = SortBy[RootReduce[FullSimplify[RT[[2]][Cels[[ci]]]]], Less];
 notmember2[nc, N[mapc2]];
 If[output,
  Cels = Join[Cels, {mapc2}]
 ];
 mapc3 = SortBy[RootReduce[FullSimplify[RT[[3]][Cels[[ci]]]]], Less];
 notmember2[nc, N[mapc3]];
 If[output,
  Cels = Join[Cels, {mapc3}]
 ];
 mapc4 = SortBy[RootReduce[FullSimplify[RT[[4]][Cels[[ci]]]]], Less];
 notmember2[nc, N[mapc4]];
 If[output,
  Cels = Join[Cels, {mapc4}]
 ];
 ci++
]
```

```
Style["The following is the number of Cells contained in our polytope",
 16, Italic, Bold, Blue, FontFamily → "Courier", LineSpacing → {1, 0}]
Length[Cels] (* This counts the number of Cells in our polytope *)
Cels;
```

Out[∘]= *The following is the number of vertices contained in our first cell C_1*

Out[∘]= 4

Out[∘]= *The following is the number of Cells contained in our polytope*

Out[∘]= 5

```
(* We have determined the sets of sets of points that define edges,
faces or cells *)
(* We will now determine the incidence lists for these different sets,
so that we know which edges belong to which faces, etc. *)
Clear[ListVertexInEdge, ListEdgeInFace, ListFaceInCel]
ListVertexInEdge = {};        (* initialize the  incidence sets*)
ListVertexInFace = {};
ListVertexInCell = {};
ListEdgeInFace = {};
ListFaceInCel = {};

NVertices = N[Vertices];
NEdges = N[Edges];
NFaces = N[Faces];
NCels = N[Cels];

For[i = 1, i ≤ Length[Edges], i++,
 (* In this for-loop we determine which vertices are in each edge,
 the vertices are identified by numbers *)
 Local = {};
 For[j = 1, j ≤ Length[Vertices], j++,
  For[l = 1, l ≤ Length[Edges[[1]]], l++,
   If[Norm[NEdges[[i, l]] - NVertices[[j]]] < 0.01,
     AppendTo[Local, j]
    ];
  ]
 ];
 AppendTo[ListVertexInEdge, Local];
]

For[i = 1, i ≤ Length[Faces], i++,
 (* In this for-loop we determine which vertices are in each face,
 the vertices are identified by numbers *)
 Local = {};
 For[j = 1, j ≤ Length[Vertices], j++,
  For[l = 1, l ≤ Length[Faces[[i]]], l++,
   If[Norm[NFaces[[i, l]] - NVertices[[j]]] < 0.01,
     AppendTo[Local, j]
    ];
  ]
 ];
 AppendTo[ListVertexInFace, Local];
]

For[i = 1, i ≤ Length[Cels], i++,
 (* In this for-loop we determine which vertices are in each cell,
 the vertices are identified by numbers *)
 Local = {};
 For[j = 1, j ≤ Length[Vertices], j++,
  For[l = 1, l ≤ Length[Cels[[1]]], l++,
   If[Norm[NCels[[i, l]] - NVertices[[j]]] < 0.01,
     AppendTo[Local, j]
    ];
  ]
 ];
 AppendTo[ListVertexInCell, Local];
]

For[i = 1, i ≤ Length[Faces],
 (* In this for-loop we determine which edges are in each face,
 the vertices are identified by numbers *)
 Local = {};
 For[j = 1, j ≤ Length[Edges],
  If[SubsetQ[ListVertexInFace[[i]], ListVertexInEdge[[j]]],
   (* This checks whether some edge (set of vertices)
    is a subset of a face (bigger set of vertices) *)
   AppendTo[Local, j]                                          (* If the
    edge is a subset of the face then we add the edgenumber to the local list *)
  ];
  j++
 ];
 AppendTo[ListEdgeInFace, Local];
 (* The local set represents a face, it is a set of edgenumbers*)
 i++
]

For[i = 1, i ≤ Length[Cels],
 (* In this for-loop we determine which edges are in each face,
 the vertices are identified by numbers *)
 Local = {};
 For[j = 1, j ≤ Length[Faces],
  If[SubsetQ[ListVertexInCell[[i]], ListVertexInFace[[j]]],
   (* This checks whether some face (set of vertices)
    is a subset of a cell (bigger set of vertices) *)
   AppendTo[Local, j]                                          (* If the
    face is a subset of the cell then we add the facenumber to the local list *)
  ];
  j++
 ];
 AppendTo[ListFaceInCel, Local];
 (* The local set represents a cell, it is a set of facenumbers*)
 i++
]

maxVert = Count[ListVertexInEdge, 1, {2}];
```

```
minVert = Count[ListVertexInEdge, 1, {2}];

(* In this for-loop we count the number of times every vertex occurs in an edge and
  check if this number is equal for every vertex, it should be the same *)
For[i = 2, i < Length[Vertices], local = Count[ListVertexInEdge, i, {2}];
 If[maxVert < local, maxVert = local, If[minVert > local, minVert = local]], i++]
If[maxVert == minVert, countVert = maxVert,
  countVert = "not an equal number for every vertex"];
Style["The following is the number of edges adjacent to each vertex",
 16, Italic, Bold, Blue, FontFamily → "Courier", LineSpacing → {1, 0}]
countVert

maxVert = Count[ListVertexInFace, 1, {2}];
minVert = Count[ListVertexInFace, 1, {2}];

(* In this for-loop we count the number of times every vertex occurs in a face and
  check if this number is equal for every vertex, it should be the same *)
For[i = 2, i < Length[Vertices], local = Count[ListVertexInFace, i, {2}];
 If[maxVert < local, maxVert = local, If[minVert > local, minVert = local]], i++]
If[maxVert == minVert, countVert = maxVert,
  countVert = "not an equal number for every vertex"];
Style["The following is the number of faces incident to each vertex",
 16, Italic, Bold, Blue, FontFamily → "Courier", LineSpacing → {1, 0}]
countVert

maxVert = Count[ListVertexInCell, 1, {2}];
minVert = Count[ListVertexInCell, 1, {2}];

(* In this for-loop we count the number of times every vertex occurs in a cell and
  check if this number is equal for every vertex, it should be the same *)
For[i = 2, i < Length[Vertices], local = Count[ListVertexInCell, i, {2}];
 If[maxVert < local, maxVert = local, If[minVert > local, minVert = local]], i++]
If[maxVert == minVert, countVert = maxVert,
  countVert = "not an equal number for every vertex"];
Style["The following is the number of cells incident to each vertex",
 16, Italic, Bold, Blue, FontFamily → "Courier", LineSpacing → {1, 0}]
countVert

maxEdge = Count[ListEdgeInFace, 1, {2}];
minEdge = Count[ListEdgeInFace, 1, {2}];
             (* In this for-loop we count the number of times every edge occurs in
  a face and check if this number is equal for every edge, it should be the same *)
For[i = 2, i < Length[Edges], local = Count[ListEdgeInFace, i, {2}];
 If[maxEdge < local, maxEdge = local, If[minEdge > local, minEdge = local]], i++]
If[maxEdge == minEdge, countEdge = maxEdge,
  countEdge = "not an equal number for every edge"];
Style["The following is the number of faces incident to each edge",
 16, Italic, Bold, Blue, FontFamily → "Courier", LineSpacing → {1, 0}]
countEdge

maxFace = Count[ListFaceInCel, 1, {2}];
minFace = Count[ListFaceInCel, 1, {2}];
             (* In this for-loop we count the number of times every face occurs in
  a cell and check if this number is equal for every face, it should be the same *)
If[k == 1 || k == 4,
 For[i = 2, i < Length[Faces], local = Count[ListFaceInCel, i, {2}];
  If[maxFace < local, maxFace = local, If[minFace > local, minFace = local]], i++],
 If[k == 3 || k == 2,
  For[i = 2, i < Length[Faces1], local = Count[ListFaceInCel, i, {2}];
   If[maxFace < local, maxFace = local, If[minFace > local, minFace = local]], i++]
 ]
]
If[maxFace == minFace, countFace = maxFace,
  countFace = "not an equal number for every face"];
Style["The following is the number of cells incident to each face",
 16, Italic, Bold, Blue, FontFamily → "Courier", LineSpacing → {1, 0}]
countFace

ListVertexInEdge;             (* To check any of the lists,
remove the ';' at the end and evaluate this mathematica-cell again *)
ListVertexInFace;
ListVertexInCell;
ListEdgeInFace;
ListFaceInCel;
```

```
Out[ ]= The following is the number of edges adjacent to each vertex

Out[ ]= 4

Out[ ]= The following is the number of faces incident to each vertex

Out[ ]= 6

Out[ ]= The following is the number of cells incident to each vertex

Out[ ]= 4

Out[ ]= The following is the number of faces incident to each edge

Out[ ]= 3

Out[ ]= The following is the number of cells incident to each face

Out[ ]= 2
```

```
(* Now that we have found all the coordinates of
 the polytope project these coordinates onto a hyperplane,
we have choosen the hyperplane perpendicular to "normal" *)
normal = {-1, -2, -2, -1};
(* The hyperplane has basis vectors w_1,w_2,w_3 which will now be determined *)
w1 = {1, aa, bb, cc};
w2 = {dd, 1, ee, ff};
w3 = {gg, hh, 1, ii};
          (* The following finds values for the variables aa,
bb,...,ii such that the vectors w1,w2,
w3 are orthogonal to each other and to the normal vector *)
s3 = FindInstance[w1.normal == 0 && w2.normal == 0 && w3.normal == 0 &&
    w1.w2 == 0 && w1.w3 == 0 && w2.w3 == 0, {aa, bb, cc, dd, ee, ff, gg, hh, ii}];
Ws = Join[{w1}, {w2}, {w3}] /. s3;
Style[
 "The following is the matrix W with row vectors that span the projection hyperplane",
 16, Italic, Bold, Blue, FontFamily → "Courier", LineSpacing → {1, 0}]
W = Replace[Ws, {x_List} :→ x, {0, -3}] (* For the ease of coding
  we use the vectors w_1,w_2,w_3 as row vectors of a matrix W *)
```

Out[ ]= ***The following is the matrix W with row vectors that span the projection hyperplane***

Out[ ]= $\left\{\left\{1, -\frac{2}{9}, -\frac{2}{9}, -\frac{1}{9}\right\}, \left\{0, 1, -\frac{4}{5}, -\frac{2}{5}\right\}, \{0, 0, 1, -2\}\right\}$

```
(* Using the normal vector of the projection hyperplane
 we can compute the projection of all coordinates  *)
ClearAll[P]
normal = Normalize[normal];
tr = {1, 2, 2, 1}; (* This is the translation vector *)
Project[v_] := v - Dot[v, normal] * normal + Dot[tr, normal] * normal;
P = FullSimplify[Map[Project, {Vertices[[1]]}]];
(* In the for loop we project each vertex of the polytope onto the hyperplane *)
For[i = 2, i ≤ Length[Vertices], i++,
  P = Join[P, FullSimplify[Map[Project, {Vertices[[i]]}]]]
 ];
Style[
 "The following is the number of vertices projected on the projection hyperplane",
 16, Italic, Bold, Blue, FontFamily → "Courier", LineSpacing → {1, 0}]
Length[P]
P;
```

Out[ ]= ***The following is the number of vertices projected on the projection hyperplane***

Out[ ]= 5

In[∘]:=
```
(* All the projected vertices lie in the chosen hyperplane,
thus they can be written as a linear combination of w_1,
w_2,w_3 using three coefficients *)
(* These three coefficients for each vertex
 will become the 3D coordinates of that vertex *)
ClearAll[H]
H = {FullSimplify[{(P[[1]].W[[1]]) / Norm[W[[1]]],
     (P[[1]].W[[2]]) / Norm[W[[2]]], (P[[1]].W[[3]]) / Norm[W[[3]]]}]}
(* In the for loop we rewrite the coordinates for every projected vertex *)
For[i = 2, i ≤ Length[P], i++,
   H = Join[H, {FullSimplify[{(P[[i]].W[[1]]) / Norm[W[[1]]],
        (P[[i]].W[[2]]) / Norm[W[[2]]], (P[[i]].W[[3]]) / Norm[W[[3]]]}]}]
 ];
Style[
 "The following is the number of vertices in the hyperplane in rewritten coordinates",
 16, Italic, Bold, Blue, FontFamily → "Courier", LineSpacing → {1, 0}]
Length[H]
H;
```

Out[∘]= $\left\{\left\{\frac{1}{90}\left(3+27\sqrt{10}-2\sqrt{15}+2\sqrt{30}\right), \frac{1}{45}\left(3\sqrt{2}-5\sqrt{15}-2\sqrt{30}\right), \frac{\sqrt{2}}{5}+\frac{1}{\sqrt{30}}\right\}\right\}$

Out[∘]= ***The following is the number of vertices***
in the hyperplane in rewritten coordinates

Out[∘]= 5

In[∘]:=
```
(* In the following lines of code we prepare some
 numbers and the rewritten coordinates of our projected vertices
 to correctly export them into a by POVray readable file *)
whetherF2exists = 0;
lengthfaces = Length[Faces];
If[k == 2 || k == 3, whetherF2exists = 1;
   lengthfaces = {Length[Faces1], Length[Faces2]}];
floats = Flatten[{whetherF2exists, facelength, Length[ListFaceInCel[[1]]],
    Length[Vertices], Length[Edges], lengthfaces, Length[Cels]}]
floattxt = ExportString[floats, "Table", "FieldSeparators" -> ","];
floatTXT = StringReplace[floattxt, {EndOfLine → ","}];
Export[FileNameJoin[{directory, "Floats.pov"}], floatTXT, "Text"];

txt = ExportString[N[H], "Table", "FieldSeparators" -> ","];
pov = StringReplace[txt, {StartOfLine → "<", EndOfLine → ">,"}];
Export[FileNameJoin[{directory, "Vectoren.pov"}], pov, "Text"];

Vtxt = ExportString[ListVertexInEdge, "Table", "FieldSeparators" -> ","];
VTXT = StringReplace[Vtxt, {EndOfLine → ","}];
Export[FileNameJoin[{directory, "VTXT.pov"}], VTXT, "Text"];
Etxt = ExportString[ListEdgeInFace, "Table", "FieldSeparators" -> ","];
ETXT = StringReplace[Etxt, {EndOfLine → ","}];
Export[FileNameJoin[{directory, "ETXT.pov"}], ETXT, "Text"];
Ftxt = ExportString[ListFaceInCel, "Table", "FieldSeparators" -> ","];
FTXT = StringReplace[Ftxt, {EndOfLine → ","}];
Export[FileNameJoin[{directory, "FTXT.pov"}], FTXT, "Text"];
```

Out[∘]= {0, 3, 4, 5, 10, 10, 5}

Bibliography

[1] Chris Cason, Thorsten Froehlich, and Christoph Lipka. *POV-Ray - The Persistence of Vision Raytracer - http://www.povray.org/*. Version 3.7, 2013.

[2] H S M Coxeter. *Regular polytopes LK - https://tue.on.worldcat.org/oclc/989538283*. Dover Publications, New York SE - XIII, 321 p., 3rd ed. edition, 1973.

[3] H S M Coxeter. *Regular complex polytopes LK - https://tue.on.worldcat.org/oclc/21562167*. Cambridge University Press, Cambridge [England] ; SE - xiv, 210 pages : illustrations ; 26 x 28 cm, 2nd ed. edition, 1991.

[4] Jeremy John Gray. *Bernhard Riemann - https://www.britannica.com/biography/Bernhard-Riemann*. Encyclopedia Brittanica, Acces date: December 24, 2020, 2020.

[5] Yanting Ji. Linear Algebra II , 2017. 2019.

[6] Akshat Mahajan. *Optimization in engineering design - http://link.springer.com/10.1007/978-3-319-56769-3*, volume 9 of *Springer Optimization and Its Applications*. Springer International Publishing, Cham, 1972.

[7] Akshat Mahajan. *(1) What is Euclidean space https://www.quora.com/What-is-Euclidean-space-and-how-is-it-related-to-a-vector-space-in-layman-terms*. 1Quora, Acces date: December 23, 2020, 2015.

[8] SchulteEgon McMullenPeter. Abstract Regular Polytopes - Peter McMullen, Egon Schulte - Google Books - https://books.google.nl/books?hl=nllr=id=JfmlMYe6MJgCoi=fndpg=PP19dq=abstract+regular+polytopesots=IIart7wsXBsig=liF3rbTomjbzXt6BmMJY6WEjxncv=onepageq=abstract regular polytopesf=false, 2003.

[9] J J O'Connor and E F Robertson. *Lüdwig Schlafli - https://mathshistory.st-andrews.ac.uk/Biographies/Schlafli/*. MacTutor, (accessed: 24.12.2020, last update: July 2007), 2007.

[10] Daniel Weller Tuesday. Lecture 16: Linear algebra with matrices. Technical report, 2019.

[11] Spencer Whitehead. Classifying Regular Polyhedra and Polytopes using Wythoff's Construction. pages 1–11, 2020.

[12] Wolfram. *Wolfram Mathematica: Modern Technical Computing - https://www.wolfram.com/mathematica/*. Wolfram, Version 11.3, 2019.