BRNO UNIVERSITY OF TECHNOLOGY
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

FACULTY OF MECHANICAL ENGINEERING
FAKULTA STROJNÍHO INŽENÝRSTVÍ

INSTITUTE OF MATHEMATICS
ÚSTAV MATEMATIKY

GEOMETRIC ALGEBRA APPLICATIONS
APLIKACE GEOMETRICKÝCH ALGEBER

MASTER'S THESIS
DIPLOMOVÁ PRÁCE

AUTHOR — **Bc. Lukáš Machálek**
AUTOR PRÁCE

SUPERVISOR — **doc. Mgr. Petr Vašík, Ph.D.**
VEDOUCÍ PRÁCE

BRNO 2021

BRNO UNIVERSITY OF TECHNOLOGY
FACULTY OF MECHANICAL ENGINEERING

Assignment Master's Thesis

Institut:	Institute of Mathematics
Student:	**Bc. Lukáš Machálek**
Degree programm:	Applied Sciences in Engineering
Branch:	Mathematical Engineering
Supervisor:	**doc. Mgr. Petr Vašík, Ph.D.**
Academic year:	2020/21

As provided for by the Act No. 111/98 Coll. on higher education institutions and the BUT Study and Examination Regulations, the director of the Institute hereby assigns the following topic of Master's Thesis:

Geometric algebra applications

Brief Description:

Geometric algebra (GA) is a powerful tool for applications in various research areas, such as mechanics and image processing. Due to geometric interpretation of GA elements, it is possible to check the algorithm correctness immediately in a software like Python. Thus the application in geometric oriented tasks is quite appropriate.

Master's Thesis goals:

Elaborating an example of GA application together with its software implementation.

Recommended bibliography:

DORST, Leendert, FONTIJNE, Daniel H. F. a MANN, Stephen. Geometric algebra for computer science: an object-oriented approach to geometry. Rev. ed. Burlington, Mass.: Morgan Kaufmann Publishers, c2007. Morgan Kaufmann series in computer graphics. ISBN 978-0-12-374942-0.

HILDENBRAND, Dietmar. Foundations of geometric algebra computing. Geometry and computing, 8. ISBN 3642317936.

PERWASS, Christian. Geometric algebra with applications in engineering. Berlin: Springer, c2009. ISBN 354089067X.

HRDINA, Jaroslav, NÁVRAT, Aleš a VAŠÍK, Petr. Geometric Algebra for Conics. Advances in Applied Clifford Algebras [online]. 2018, 28(3) [cit. 2019-06-28]. DOI: 10.1007/s00006-018-0879-2. ISSN 0188-7009. Dostupné z: http://link.springer.com/10.1007/s00006-018-0879-2.

Deadline for submission Master's Thesis is given by the Schedule of the Academic year 2020/21

In Brno,

L. S.

prof. RNDr. Josef Šlapal, CSc. Director of the Institute	doc. Ing. Jaroslav Katolický, Ph.D. FME dean

Summary
This diploma thesis deals with geometric algebra for conics (GAC) in autonomous navigation, presented on robot movement in a tube. First, the theoretical concepts are introduced. Consequently, the representations of conics in GAC are presented. Then an engine is implemented, which is capable of performing basic operations in GAC including displaying conics, which are entered in GAC context. In the end an algorithm is presented, which estimates the tube axis using points, placed into space from image, where we place center of an ellipse, which is obtained by image filter and fitting algorothm.

Abstrakt
Tato diplomová práce se zabývá využitím geometrické algebry pro kuželosečky (GAC) v autonomní navigaci, prezentované na pohybu robota v trubici. Nejprve jsou zavedeny teoretické pojmy z geometrických algeber. Následně jsou prezentovány kuželosečky v GAC. Dále je provedena implementace enginu, který je schopný provádět základní operace v GAC, včetně zobrazování kuželoseček zadaných v kontextu GAC. Nakonec je ukázán algoritmus, který odhadne osu trubice pomocí bodů, které umístí do prostoru pomocí středů elips, umístěných v obrazu, získaných obrazovým filtrem a fitovacím algoritmem.

Keywords
geometric algebra, geometric algebra for conics, computer graphics, projective geometry, image filters, autonomous navigation, Csharp, Unity engine

Klíčová slova
geometrická algebra, geometrická algebra pro kuželosečky, počítačová grafika, projektivní geometrie, obrazové filtry, autonomní navigace, Csharp, Unity engine

MACHÁLEK, L. *Geometric algebra applications*. Brno: Vysoké učení technické v Brně, Faculty of Mechanical Engineering, 2021. 61 s. Vedoucí doc. Mgr. Petr Vašík, Ph.D.

I declare that I have written the diploma thesis *Geometric Algebra Applications* on my own under the guidance of my supervisor doc. Mgr. Petr Vašík, Ph.D., and that I used the sources listed in references.

Bc. Lukáš Machálek

I would like to express my sincerest thanks to the supervisor of my diploma thesis, doc. Mgr. Petr Vašík, Ph.D. for his valuable advices, useful comments, and the time he spent helping me. I also want to thank my family, especially my parents for supporting me.

Bc. Lukáš Machálek

Contents

Introduction

Autonomous navigation is a very up to date topic. There are a lot of sensors that provide specific data, that can be used to analyze environment and use it for trajectory planning. One of the most used sensors are cameras, which capture the environment as an image from camera's perspective. Sophisticated algorithms then make an estimation about the environment in front of that camera. But extraction of details for orientation from a single picture is very difficult without any information about the environment. Before the software can be developed, simulations are necessary. First step can be simple example of that environment, which can be clearly recognizable by the algorithms, because it has direct signs. Autonomous navigation of a robot with camera in a tube can be initialized with simulation, which is described by this thesis. Easy case to solve is just a straight tube. The robot with a camera and light source can simply be navigated to the direction of the darkest point in an image, given by the camera. In a case of a curved tube, the same approach can lead to a collision with the tube surface. Safe path would follow the tube axis.

Goal of this thesis is to find a conic that follows elliptic contour, that can be seen in a curved tube (Figure 2.2) and make path using the center of the conic. We create a testing tube and develop an algorithm, the first part of which generates such conic. The second part extracts point in space as the center of that conic. The environment was created with Unity game engine, which create 2D or 3D environment with many useful tools. Unity uses C# as scripting programming language. Conic sections and manipulation with them can be defined using geometric algebras.

Usage of geometric algebras in the computer science is interesting for many reasons, see [3, 5]. The main advantage comes with geometrically intuitive development of algorithms. It connects standard vector space calculus with many mathematical tools. Christian Perwass [10] examines all aspects essential for a successful application of geometric algebra. Our focus in this thesis is placed on Geometric Algebra for Conics (GAC), see [6, 7]. It can be described as 256-dimensional algebra, which contains conformal embedding of Euclidean space $\mathbb{R}^2$, together with representations of conics and Euclidean transformations. Because working with such high dimensional space can be demanding in terms of computational time, it is reasonable to look for optimization. We put our interest on effective computing of geometric algebra's operation called geometric product, because this operation can take a lot of time with large dimension of the algebra.

This thesis is divided into 2 parts. Chapter 1 gives a theoretical background to geometric algebras, more specifically to $\mathbb{G}_{5,3}$. As in majority of the thesis we work with properties and basic operations of the geometric algebra, in Section 1.1.1 we introduce general geometric algebra concept. In Section 1.2 we present GAC together with representations of conics, Euclidean transformations and conic fitting algorithm. Software, developed for this thesis, is able to perform computations presented in Section 1.2 and display graphical result.

In Chapter 2 we present the mentioned software. As this application does not use preset objects in Unity, we present how to generate a tube in computer graphic. Then we introduce simple algorithms for point selection, the purpose of which is to highlight a tube elliptic contour (when the camera is moving through curve). Finally, with the tube radius and projective geometry in computer graphics together with the properties of

fitted ellipse we estimate the axis. The program is created with Unity 2019.4.18f1, scripts are written with C# 9.0 and MATLAB R2020b.

1. Theoretic introduction

1.1. Geometric Algebra

We begin this Section with defining a vector space $\mathbb{R}^{p,q}$ equipped with a bilinear form. Then we continue with axiomatic definition of geometric algebra. We observe properties of its operation on basis vectors. We define specific way how basis of the geometric algebra is ordered. We also introduce another two important operations and we finish this section with the duality property. We follow [10], where the general concept of geometric algebra is introduced in a detailed way.

1.1.1. Vector Space $\mathbb{R}^{p,q}$

The following text assumes knowledge of the concept of vector spaces, linear forms, basic concept of an algebra. Let $\mathbb{R}^{p,q}$ denote a $(p+q)$-dimensional vector space over the set of real numbers $\mathbb{R}$. Furthermore, let f be a commutative bilinear form[1] on this space. The reason, why we use notation p, q, comes from the property on basis vectors of the space $\mathbb{R}^{p,q}$ together with f defined as follows.

Definition 1.1.1. The canonical basis of $\mathbb{R}^{p,q}$ denoted by $\overline{\mathbb{R}}^{p,q}$ is defined as the totally ordered set

$$\overline{\mathbb{R}}^{p,q} = (\boldsymbol{e}_1, \ldots, \boldsymbol{e}_p, \boldsymbol{e}_{p+1}, \ldots, \boldsymbol{e}_{p+q}) \subset \mathbb{R}^{p,q}, \tag{1.1}$$

where $\boldsymbol{e}_i$ has the following property

$$f(\boldsymbol{e}_i, \boldsymbol{e}_j) = \begin{cases} 1, & 1 \leq i = j \leq p, \\ -1, & p+1 \leq i = j \leq p+q, \\ 0, & i \neq j. \end{cases} \tag{1.2}$$

This bilinear form is understood to be a scalar product in the context of geometric algebras, however without property $f(\boldsymbol{u}, \boldsymbol{u}) > 0$. Thus we call the bilinear form (1.2) a pseudoscalar product in the vector spaces context. Let the basis elements of $\overline{\mathbb{R}}^{p,q}$ be such that $\boldsymbol{e}_1, \ldots, \boldsymbol{e}_p$ have positive signature and $\boldsymbol{e}_{p+1}, \ldots, \boldsymbol{e}_{p+q}$ have negative signature, i.e. $f(\boldsymbol{e}_i, \boldsymbol{e}_i) = 1\,(-1)$ for $i = 1, \ldots, p$ $(i = p+1, \ldots, p+q)$. The space $\mathbb{R}^{p,q}$ is understood to be the vector space $\mathbb{R}^{p+q}$ with the pseudoscalar product (1.2).

Definition 1.1.2. Let r be a quadratic from associated with the pseudoscalar product f. That is, for $\boldsymbol{u} \in \mathbb{R}^{p+q}$, $r(\boldsymbol{u}) = f(\boldsymbol{u}, \boldsymbol{u})$. Then the vector space $\mathbb{R}^{p+q}$ together with quadratic form r create a quadratic space $(\mathbb{R}^{p,q}, r)$.

1.1.2. Axiomatic Definition

Definition 1.1.3. Let $\mathbb{A}(\mathbb{R}^{p,q})$ denote an associative algebra over the quadratic space $(\mathbb{R}^{p,q}, r)$. Let a symbol $*$ denote an algebraic product. The Algebra $\mathbb{A}(\mathbb{R}^{p,q})$ is said to be a geometric algebra if for every vector $\boldsymbol{u} \in \mathbb{R}^{p,q} \subset \mathbb{A}(\mathbb{R}^{p,q})$, $\boldsymbol{u} * \boldsymbol{u} = r(\boldsymbol{u})$ holds.

[1] If we would further assume that $f(\boldsymbol{u}, \boldsymbol{u}) > 0$ we would end up with a scalar product.

The geometric algebra over the vector space $\mathbb{R}^{p,q}$ is denoted by $\mathbb{G}(\mathbb{R}^{p,q})$ or simply $\mathbb{G}_{p,q}$. The algebraic product is called a geometric product. The elements of $\mathbb{G}_{p,q}$ are called multivectors. Now we give an axiomatic definition of the multivectors.

$$\begin{aligned} &(i) \quad \forall \boldsymbol{U}, \boldsymbol{V} \in \mathbb{G}_{p,q}, \quad \exists \boldsymbol{W} \in \mathbb{G}_{p,q} : \quad \boldsymbol{W} = \boldsymbol{U} + \boldsymbol{V}, \\ &(ii) \quad \forall \boldsymbol{U} \in \mathbb{G}_{p,q}, \quad \forall c \in \mathbb{R} : \quad \exists c\boldsymbol{U} \in \mathbb{G}_{p,q} \end{aligned} \tag{1.3}$$

The property (i) reads that $\mathbb{G}_{p,q}$ is closed under addition, and (ii) reads that $\mathbb{G}_{p,q}$ is closed under multiplication by constant. Note that the filed of real numbers $\mathbb{R} \subset \mathbb{G}_{p,q}$. Multivectors also satisfy conditions for vectors in the vector space over the $\mathbb{R}$.

$$\begin{aligned} &(i) \quad \forall \boldsymbol{U}, \boldsymbol{V}, \boldsymbol{W} \in \mathbb{G}_{p,q} : \quad (\boldsymbol{U} + \boldsymbol{V}) + \boldsymbol{W} = \boldsymbol{U} + (\boldsymbol{V} + \boldsymbol{W}), \\ &(ii) \quad \forall \boldsymbol{U}, \boldsymbol{V} \in \mathbb{G}_{p,q} : \quad \boldsymbol{U} + \boldsymbol{V} = \boldsymbol{V} + \boldsymbol{U}, \\ &(iii) \quad \exists 0 \in \mathbb{G}_{p,q}, \quad \forall \boldsymbol{U} \in \mathbb{G}_{p,q} : \quad \boldsymbol{U} + 0 = \boldsymbol{U}, \\ &(iv) \quad \forall \boldsymbol{U} \in \mathbb{G}_{p,q}, \quad \exists -\boldsymbol{U} \in \mathbb{G}_{p,q} : \quad \boldsymbol{U} + (-\boldsymbol{U}) = 0, \\ &(v) \quad \forall \boldsymbol{U}, \boldsymbol{V} \in \mathbb{G}_{p,q}, \quad \forall c \in \mathbb{R} : \quad c(\boldsymbol{U} + \boldsymbol{V}) = c\boldsymbol{U} + c\boldsymbol{V}, \\ &(vi) \quad \forall \boldsymbol{U} \in \mathbb{G}_{p,q}, \quad \forall c, d \in \mathbb{R} : \quad (c + d)\boldsymbol{U} = c\boldsymbol{U} + d\boldsymbol{U}, \\ &(vii) \quad \forall \boldsymbol{U} \in \mathbb{G}_{p,q}, \quad \forall a, b \in \mathbb{R} : \quad (ab)\boldsymbol{U} = a(b\boldsymbol{U}), \\ &(viii) \quad \text{for } 1 \in \mathbb{R}, \quad \forall \boldsymbol{U} \in \mathbb{G}_{p,q} : \quad 1\boldsymbol{U} = \boldsymbol{U}. \end{aligned} \tag{1.4}$$

Axioms for the geometric product "$*$" are as follows:

$$\begin{aligned} &(i) \quad \forall \boldsymbol{U}, \boldsymbol{V} \in \mathbb{G}_{p,q} : \quad (\boldsymbol{U} * \boldsymbol{V}) \in \mathbb{G}_{p,q}, \\ &(ii) \quad \forall \boldsymbol{U}, \boldsymbol{V}, \boldsymbol{W} \in \mathbb{G}_{p,q} : \quad (\boldsymbol{U} * \boldsymbol{V}) * \boldsymbol{W} = \boldsymbol{U} * (\boldsymbol{V} * \boldsymbol{W}), \\ &(iii) \quad \forall \boldsymbol{U}, \boldsymbol{V}, \boldsymbol{W} \in \mathbb{G}_{p,q} : \quad \boldsymbol{W} * (\boldsymbol{U} + \boldsymbol{V}) = \boldsymbol{W} * \boldsymbol{U} + \boldsymbol{W} * \boldsymbol{V}, \\ &\qquad\qquad\qquad\qquad\qquad\qquad (\boldsymbol{V} + \boldsymbol{W}) * \boldsymbol{U} = \boldsymbol{V} * \boldsymbol{U} + \boldsymbol{W} * \boldsymbol{U}, \\ &(iv) \quad \forall \boldsymbol{U} \in \mathbb{G}_{p,q}, \quad \forall c \in \mathbb{R} : \quad c * \boldsymbol{U} = \boldsymbol{U} * c = c\boldsymbol{U}. \end{aligned} \tag{1.5}$$

All the axioms given so far define an associative algebra. What actually separates the Geometric Algebra from other algebras is the defining equation.

$$\forall \boldsymbol{u} \in \mathbb{R}^{p,q} \subset \mathbb{G}_{p,q} \quad \boldsymbol{u} * \boldsymbol{u} = r(\boldsymbol{u}, \boldsymbol{u}) \quad \in \mathbb{R}. \tag{1.6}$$

To get some insight to this concept, lets present example on the geometric algebra $\mathbb{G}_2$. Let $u_1\boldsymbol{e}_1, u_2\boldsymbol{e}_2 \in \mathbb{R}^2 \subset \mathbb{G}_2$, then $u_1u_2\boldsymbol{e}_1 * \boldsymbol{e}_2 \in \mathbb{G}_2$. Also

$$u_1u_2(\boldsymbol{e}_1 * \boldsymbol{e}_2) * (u_2\boldsymbol{e}_2) = u_1u_2^2\boldsymbol{e}_1 * (\boldsymbol{e}_2 * \boldsymbol{e}_2) = u_1u_2^2\boldsymbol{e}_1 \in \mathbb{G}_2.$$

Now we observe some basic properties. Using the property (1.2) of pseudoscalar product f for $\boldsymbol{e}_i \in \overline{\mathbb{R}}^{p,q}$ elements and equation (1.6) we get:

$$\boldsymbol{e}_i * \boldsymbol{e}_i = \begin{cases} 1, & 1 \leq i \leq p, \\ -1, & p \leq i \leq p + q. \end{cases} \tag{1.7}$$

For any vectors $\boldsymbol{u}, \boldsymbol{v} \in \mathbb{R}^{p,q} \subset \mathbb{G}_{p,q}$ the following holds:

$$\begin{aligned} (\boldsymbol{u}+\boldsymbol{v}) * (\boldsymbol{u}+\boldsymbol{v}) &= f(\boldsymbol{u}+\boldsymbol{v}, \boldsymbol{u}+\boldsymbol{v}), \\ \boldsymbol{u}*\boldsymbol{u} + \boldsymbol{u}*\boldsymbol{v} + \boldsymbol{v}*\boldsymbol{u} + \boldsymbol{v}*\boldsymbol{v} &= f(\boldsymbol{u},\boldsymbol{u}) + 2f(\boldsymbol{u},\boldsymbol{v}) + f(\boldsymbol{v},\boldsymbol{v}), \\ \tfrac{1}{2}(\boldsymbol{u}*\boldsymbol{v} + \boldsymbol{v}*\boldsymbol{u}) &= f(\boldsymbol{u},\boldsymbol{v}). \end{aligned} \tag{1.8}$$

Since we know that for $\boldsymbol{e}_i, \boldsymbol{e}_j \in \overline{\mathbb{R}}^{p,q} \subset \mathbb{G}_{p,q}$, $f(\boldsymbol{e}_i, \boldsymbol{e}_j) = 0$, $i \neq j$, then by using (1.8) we get:

$$\tfrac{1}{2}(\boldsymbol{e}_i * \boldsymbol{e}_j + \boldsymbol{e}_j * \boldsymbol{e}_i) = 0 \quad \iff \quad \boldsymbol{e}_i * \boldsymbol{e}_j = -\boldsymbol{e}_j * \boldsymbol{e}_i. \tag{1.9}$$

Further in the text, we omit the symbol $*$ and instead of $\boldsymbol{U} * \boldsymbol{V}$, we write just $\boldsymbol{U}\boldsymbol{V}$.

1.1.3. Basis of Geometric Algebra

In this subsection we show how to construct an algebraic basis of $\mathbb{G}_{p,q}$. Firstly, let $\mathbb{S}[i]$ be an i-th element of an ordered set $\mathbb{S}$. A product operator $\prod$ is understood to be the geometric product for multiple basis elements.

Definition 1.1.4. A basis blade of $\mathbb{G}_{p,q}$ is the geometric product of a number of different elements of the canonical basis $\overline{\mathbb{R}}^{p,q}$ of the vector space $\mathbb{R}^{p,q}$. Let $\mathbb{S} \subseteq \{1, \ldots, p+q\}$, then $\boldsymbol{e}_{\mathbb{S}}$ denotes the basis blade:

$$\boldsymbol{e}_{\mathbb{S}} = \boldsymbol{e}_{\mathbb{S}[1]} \ldots \boldsymbol{e}_{\mathbb{S}[|\mathbb{S}|]} = \prod_{i=1}^{|\mathbb{S}|} \overline{\mathbb{R}}^{p,q}[\mathbb{S}[i]], \tag{1.10}$$

where the number $|\mathbb{S}|$ denotes the size[2] of a set $\mathbb{S}$ and $\overline{\mathbb{R}}^{p,q}[\mathbb{S}[i]]$ is the $\mathbb{S}[i]$-th element of the canonical basis $\overline{\mathbb{R}}^{p,q}$. Also $\boldsymbol{e}_{\emptyset} = 1$ is the basis blade of $\mathbb{G}_{p,q}$.

Example 1.1.1. For example if we use geometric algebra $\mathbb{G}_{5,3}$ and set $\mathbb{S} = \{1, 3, 6, 7\}$, then

$$\boldsymbol{e}_{\mathbb{S}} = \prod_{i=1}^{4} \overline{\mathbb{R}}^{5,3}[\mathbb{S}[i]] = \overline{\mathbb{R}}^{5,3}[1]\, \overline{\mathbb{R}}^{5,3}[3]\, \overline{\mathbb{R}}^{5,3}[6]\, \overline{\mathbb{R}}^{5,3}[7] = \boldsymbol{e}_1\boldsymbol{e}_3\boldsymbol{e}_6\boldsymbol{e}_7.$$

Definition 1.1.5. The grade of a basis blade $\boldsymbol{e}_{\mathbb{S}} \in \mathbb{G}_{p,q}$, where $\mathbb{S} \subseteq \{1, \ldots, p+q\}$ is denoted as $\mathrm{gr}(\boldsymbol{e}_{\mathbb{S}})$ and is defined as $\mathrm{gr}(\boldsymbol{e}_{\mathbb{S}}) = |\mathbb{S}|$. The grade of $\boldsymbol{e}_{\emptyset}$ is $\mathrm{gr}(1) = 0$.

In the vector space $\mathbb{R}^{p,q}$ with canonical basis $\overline{\mathbb{R}}^{p,q}$ there are 2^{p+q} ways how to uniquely multiply the basis vectors of $\overline{\mathbb{R}}^{p,q}$. In other words, in the Geometric algebra $\mathbb{G}_{p,q}$ there exist 2^{p+q} linearly independent basis blades.

Recall that the geometric product is associative. So we can write $(\boldsymbol{e}_i\boldsymbol{e}_j)\boldsymbol{e}_k$ as $\boldsymbol{e}_i\boldsymbol{e}_j\boldsymbol{e}_k$, $i, j, k \in \{1, \ldots, p+q\}$ and so on with more elements. We know that $\boldsymbol{e}_i\boldsymbol{e}_j = -\boldsymbol{e}_j\boldsymbol{e}_i$ if $i \neq j$. So if we switch the positions of two basis blades, the only thing changed is the signature. But this leads to a number of possible bases for one geometric algebra $\mathbb{G}_{p,q}$. On the other hand these bases are always isomorphic. Clearly, every basis of the same geometric algebra $\mathbb{G}_{p,q}$ must have the same number of elements, and there is an option to switch positions of the basis vectors. Further in this subsection an algorithm how to order the basis elements is described.

[2] In other words, the number of elements of the set, i.e. cardinality.

It is good to make an ordered set $\mathbb{A}$ for practical reasons, because it can give a formula how to create the canonical basis for a general geometric algebra $\mathbb{G}_{p,q}$. Firstly, the elements are ordered by their grade, then indexes of the base vectors are ordered in ascending sequence. This approach make the implementation of following concepts easier. For this purpose we can use totaly ordered power set.

Definition 1.1.6. Let $\mathbb{I} = \{1, \ldots, p+q\} \subset \mathbb{N}$. Clearly $|\mathbb{I}| = p+q$. Denote by $\mathcal{P}(\mathbb{I})$ the power set of $\mathbb{I}$, i. e. the cardinality of $\mathcal{P}(\mathbb{I})$ is 2^{p+q}. The ordered power set of $\mathbb{I}$, denoted by $\mathcal{P}_O(\mathbb{I})$, is a totally ordered set with the following properties. The elements of $\mathcal{P}_O(\mathbb{I})$, which are subsets of $\mathbb{I}$, are ordered by cardinality in ascending order. The members of each element of $\mathcal{P}_O(\mathbb{I})$ are ordered in ascending order. The elements of $\mathcal{P}_O(\mathbb{I})$ of equal cardinality are ordered in lexicographical order.

Example 1.1.2. Let $\mathbb{I} = \{1, 2, 3\}$, $|\mathbb{I}| = 3$, cardinality of the corresponding power set is $2^3 = 8$.

$$\mathcal{P}_O(\mathbb{I}) = \big\{\{\emptyset\}, \{1\}, \{2\}, \{3\}, \{1,2\}, \{1,3\}, \{2,3\}, \{1,2,3\}\big\}. \tag{1.11}$$

Definition 1.1.7. The canonical basis of the geometric algebra $\mathbb{G}_{p,q}$, denoted as $\overline{\mathbb{G}}_{p,q}$, is constructed as follows. Let $\overline{\mathbb{R}}^{p,q}$ be canonical basis

$$\overline{\mathbb{R}}^{p,q} = (\boldsymbol{e}_1, \ldots, \boldsymbol{e}_p, \boldsymbol{e}_{p+1}, \ldots, \boldsymbol{e}_{p+q}),$$

and let $\mathbb{I} = \{1, \ldots, p+q\}$. Canonical basis of the geometric algebra $\mathbb{G}_{p,q}$ is given by

$$\overline{\mathbb{G}}_{p,q} = \big\{\boldsymbol{e}_{\mathbb{S}} \ : \ \mathbb{S} \in \mathcal{P}_O(\mathbb{I})\big\}. \tag{1.12}$$

The order of elements in $\overline{\mathbb{G}}_{p,q}$ is analogous to $\mathcal{P}_O(\mathbb{I})$.

Notation of the basis blades $\boldsymbol{e}_i\boldsymbol{e}_j$ can be simplified as $\boldsymbol{e}_{ij}$. Let $\boldsymbol{E}_i = \overline{\mathbb{G}}_{p,q}[i]$, then every multivector $\boldsymbol{U} \in \mathbb{G}_{p,q}$ can be expressed as a linear combination of the basis blades;

$$\boldsymbol{U} = \sum_{i=1}^{2^{p+q}} u_i \boldsymbol{E}_i, \quad \boldsymbol{E}_i \in \overline{\mathbb{G}}_{p,q}, \quad u_i \in \mathbb{R}. \tag{1.13}$$

Definition 1.1.8. Let $\boldsymbol{U}, \boldsymbol{V} \in \mathbb{G}_{p,q}$, where $\boldsymbol{U} = \sum_i u_i \boldsymbol{E}_i, \boldsymbol{V} = \sum_i v_i \boldsymbol{E}_i$, $u_i, v_i \in \mathbb{R}$, then

$$\boldsymbol{U}\boldsymbol{V} = \sum_{i,j} u_i v_j \boldsymbol{E}_i \boldsymbol{E}_j. \tag{1.14}$$

Definition 1.1.9. An element with the highest grade is called a pseudoscalar and is denoted as $\boldsymbol{I}$.

$$\boldsymbol{I} = \overline{\mathbb{G}}_{p,q}[2^{p+q}] = \boldsymbol{E}_{2^{p+q}} = \boldsymbol{e}_1\boldsymbol{e}_2 \cdots \boldsymbol{e}_{p+q}. \tag{1.15}$$

1.1.4. Inner and Outer Product

This subsection provides the definition of an inner and outer product, which are two important operations in geometric algebra. They have geometrical meaning in special cases, like computing distances, angles, relations to objects or associate points to objects, similarly to the scalar and cross product of vectors in vector spaces context.

Definition 1.1.10. Let $\boldsymbol{E}_i = \overline{\mathbb{G}}_{p,q}[i]$, then the grade projection of the $\boldsymbol{E}_i$ to the grade k is denoted as $\langle \boldsymbol{E}_i \rangle_k$ and defined as follows:

$$\langle \boldsymbol{E}_i \rangle_k = \begin{cases} \boldsymbol{E}_i, \mathrm{gr}(\boldsymbol{E}_i) = k, \\ 0, \ \ \mathrm{gr}(\boldsymbol{E}_i) \neq k. \end{cases} \tag{1.16}$$

Let $\boldsymbol{U}, \boldsymbol{V} \in \mathbb{G}_{p,q}$, then their geometric product can be written, in terms of grade projection, as

$$\boldsymbol{UV} = \sum_{i=0}^{p+q} \langle \boldsymbol{UV} \rangle_i. \tag{1.17}$$

This concept of preserving the base blade only if it is a same grade as the grade of the projection is a simple way how to define each of the following operations.

Definition 1.1.11. Let $\boldsymbol{E}_i = \overline{\mathbb{G}}_{p,q}[i]$, then the inner product of the two basis blades $\boldsymbol{E}_i$, $\boldsymbol{E}_j$, with $k = \mathrm{gr}(\boldsymbol{E}_i)$ $l = \mathrm{gr}(\boldsymbol{E}_j)$ is denoted by dot and defined as

$$\boldsymbol{E}_i \cdot \boldsymbol{E}_j = \begin{cases} \langle \boldsymbol{E}_i \boldsymbol{E}_j \rangle_{|k-l|} & , \ i, j > 0, \\ 0, & i = 0 \text{ and/or } j = 0. \end{cases} \tag{1.18}$$

Definition 1.1.12. Let $\boldsymbol{E}_i = \overline{\mathbb{G}}_{p,q}[i]$, then the outer product of the two basis blades $\boldsymbol{E}_i$, and $\boldsymbol{E}_j$, with $k = \mathrm{gr}(\boldsymbol{E}_i)$ $l = \mathrm{gr}(\boldsymbol{E}_j)$ is denoted by $\wedge$ and defined as

$$\boldsymbol{E}_i \wedge \boldsymbol{E}_j = \langle \boldsymbol{E}_i \boldsymbol{E}_j \rangle_{k+l}. \tag{1.19}$$

This concept can be explained as follows. If we multiply two basis blades $\boldsymbol{E}_i$, $\boldsymbol{E}_j$ by the inner product, it results in zero if the element with the lower grade does not include the elements of the other basis blade. On the other hand, the outer product requires both basis blades to have distinct elements in each of the basis blades. Note that the outer product of two basis blades $\boldsymbol{E}_i$ a $\boldsymbol{E}_j$, with grades $k = \mathrm{gr}(\boldsymbol{E}_i)$ $l = \mathrm{gr}(\boldsymbol{E}_j)$ such that the sum $k + l$ is greater than the dimension of the vector space $\mathbb{R}^{p,q}$, is always zero, because if we just substitute into equation (1.19), we see that the geometric product of these 2 basis blades has at most grade $p + q$, so the grade projection is 0.

Example 1.1.3. The following examples show the previous explanation on elements of $\overline{\mathbb{G}}_{5,3}$.

$$\begin{aligned} &(\boldsymbol{e}_1\boldsymbol{e}_2\boldsymbol{e}_3) \cdot \boldsymbol{e}_3 = \langle \boldsymbol{e}_1\boldsymbol{e}_2\boldsymbol{e}_3\boldsymbol{e}_3 \rangle_{|3-1|} = \langle \boldsymbol{e}_1\boldsymbol{e}_2 \rangle_2 = \boldsymbol{e}_1\boldsymbol{e}_2, \\ &(\boldsymbol{e}_1\boldsymbol{e}_2) \cdot \boldsymbol{e}_3 = \langle \boldsymbol{e}_1\boldsymbol{e}_2\boldsymbol{e}_3 \rangle_{|2-1|} = \langle \boldsymbol{e}_1\boldsymbol{e}_2\boldsymbol{e}_3 \rangle_1 = 0, \\ &(\boldsymbol{e}_1\boldsymbol{e}_2\boldsymbol{e}_3) \wedge (\boldsymbol{e}_4\boldsymbol{e}_5\boldsymbol{e}_6\boldsymbol{e}_8) = \langle \boldsymbol{e}_1\boldsymbol{e}_2\boldsymbol{e}_3\boldsymbol{e}_4\boldsymbol{e}_5\boldsymbol{e}_6\boldsymbol{e}_8 \rangle_{3+4} = \boldsymbol{e}_1\boldsymbol{e}_2\boldsymbol{e}_3\boldsymbol{e}_4\boldsymbol{e}_5\boldsymbol{e}_6\boldsymbol{e}_8, \\ &(\boldsymbol{e}_1\boldsymbol{e}_2\boldsymbol{e}_3) \wedge (\boldsymbol{e}_3\boldsymbol{e}_4\boldsymbol{e}_5\boldsymbol{e}_6\boldsymbol{e}_8) = \langle \boldsymbol{e}_1\boldsymbol{e}_2\boldsymbol{e}_3\boldsymbol{e}_3\boldsymbol{e}_4\boldsymbol{e}_5\boldsymbol{e}_6\boldsymbol{e}_8 \rangle_{3+5} = \langle \boldsymbol{e}_1\boldsymbol{e}_2\boldsymbol{e}_4\boldsymbol{e}_5\boldsymbol{e}_6\boldsymbol{e}_8 \rangle_8 = 0. \end{aligned}$$

Because the inner and outer product of basis blades are either zero or the geometric product these two, it is easy to extend them on general multivectors $\boldsymbol{U}, \boldsymbol{V} \in \mathbb{G}_{p,q}$, where $\boldsymbol{U} = \sum_i u_i \boldsymbol{E}_i$ and $\boldsymbol{V} = \sum_i v_i \boldsymbol{E}_i$, $u_i, v_i \in \mathbb{R}$:

$$\boldsymbol{U} \cdot \boldsymbol{V} = \sum_{i,j} u_i v_j (\boldsymbol{E}_i \cdot \boldsymbol{E}_j), \tag{1.20}$$

$$\boldsymbol{U} \wedge \boldsymbol{V} = \sum_{i,j} u_i v_j (\boldsymbol{E}_i \wedge \boldsymbol{E}_j). \tag{1.21}$$

Definition 1.1.13. Let $\boldsymbol{u}_i \in \mathbb{R}^{p,q}$, $i = 1, \ldots, k$, $k \leq p+q$ be linearly independet vectors. Then we call outer product of these vectors as k-blade. Such blade is denoted by $\boldsymbol{U}_{\langle k \rangle}$.

1.1.5. Inversion

In this Subsection, we briefly introduce inversion with an examples on vectors $\boldsymbol{u} \in \mathbb{R}^{p,q} \subset \mathbb{G}_{p,q}$, on basis blades $\boldsymbol{E}_i \in \overline{\mathbb{G}}_{p,q}$.

Definition 1.1.14. Let $\boldsymbol{U} \in \mathbb{G}_{p,q}$ be some general multivector. Then if exists $\boldsymbol{U}^{-1} \in \mathbb{G}_{p,q}$ such that

$$\boldsymbol{U}\boldsymbol{U}^{-1} = 1, \tag{1.22}$$

then we say that multivector $\boldsymbol{U}^{-1}$ is an inverse to the multivector $\boldsymbol{U}$.

Example 1.1.4. Simple example is finding an inverse $\boldsymbol{u}^{-1} \in \mathbb{R}^{p,q} \subset \mathbb{G}_{p,q}$ to the vector $\boldsymbol{u} \in \mathbb{R}^{p,q} \subset \mathbb{G}_{p,q}$. Firstly we show why $\boldsymbol{u} \cdot \boldsymbol{u} = \boldsymbol{u}\boldsymbol{u}$:

$$\begin{aligned}
\boldsymbol{u} \cdot \boldsymbol{u} &= \sum_{i,j}^{p+q} u_i \boldsymbol{e}_i \cdot u_j \boldsymbol{e}_j = \sum_{i,j}^{p+q} u_i u_j \langle \boldsymbol{e}_i * \boldsymbol{e}_j \rangle_0 = \sum_{i}^{p+q} u_i u_i (\boldsymbol{e}_i * \boldsymbol{e}_i). \\
\boldsymbol{u} * \boldsymbol{u} &= \sum_{i,j}^{p+q} u_i \boldsymbol{e}_i * u_j \boldsymbol{e}_j = \sum_{i}^{p+q} u_i u_i (\boldsymbol{e}_i * \boldsymbol{e}_i) + \sum_{i,j\, i \neq j}^{p+q} u_i u_j (\boldsymbol{e}_i * \boldsymbol{e}_j) = \sum_{i}^{p+q} u_i u_i (\boldsymbol{e}_i * \boldsymbol{e}_i).
\end{aligned}$$

Term $\sum_{i,j\, i \neq j}^{p+q} u_i u_j (\boldsymbol{e}_i * \boldsymbol{e}_j)$ is zero because when $i \neq j$ we can find $u_i u_j (\boldsymbol{e}_i * \boldsymbol{e}_j)$ for every $u_j u_i (\boldsymbol{e}_j * \boldsymbol{e}_i)$. Since $\boldsymbol{e}_j * \boldsymbol{e}_i = -\boldsymbol{e}_i * \boldsymbol{e}_j$, all the second grade elements vanish. We had just showed that (1.6) holds. Then

$$\boldsymbol{u}^{-1} = \frac{\boldsymbol{u}}{\boldsymbol{u} \cdot \boldsymbol{u}}, \quad \boldsymbol{u} * \frac{\boldsymbol{u}}{\boldsymbol{u} \cdot \boldsymbol{u}} = \frac{\boldsymbol{u} * \boldsymbol{u}}{\boldsymbol{u} \cdot \boldsymbol{u}} = 1.$$

Definition 1.1.15. Let $\boldsymbol{E}_i \in \overline{\mathbb{G}}_{p,q}$. Then a reverse $\widetilde{\boldsymbol{E}}_i$ of the basis blade $\boldsymbol{E}_i$ is defined as

$$\widetilde{\boldsymbol{E}}_i = \prod_{j=1}^{|\mathbb{S}_i|} \boldsymbol{e}_{[\mathbb{S}_i[|\mathbb{S}_i|-j+1]]},$$

where $\mathbb{S}_i = \mathcal{P}_O(\mathbb{I})[i]$, $\mathbb{I} = \{1, \ldots, p+q\}$. Basically we reverse the order of elements in the set $\mathcal{P}_O(\mathbb{I})[i]$.

Reverse of general multivector is reverse of each basis blade in the multivector. Let $\boldsymbol{U} \in \mathbb{G}_{p,q}$, then

$$\boldsymbol{U} = \sum_{i=1}^{2^{p+q}} a_i \boldsymbol{E}_i, \quad \widetilde{\boldsymbol{U}} = \sum_{i=1}^{2^{p+q}} a_i \widetilde{\boldsymbol{E}}_i \quad \boldsymbol{E}_i \in \overline{\mathbb{G}}_{p,q}, \quad u_i \in \mathbb{R}. \tag{1.23}$$

Example 1.1.5. This Example follows [10], with a bit different approach. To see what is an inverse to the basis blade $\boldsymbol{E}_i \in \overline{\mathbb{G}}_{p,q}$, let $\widetilde{\boldsymbol{E}}_i$ be a reverse to the basis blade. Let $r \in \mathbb{N}$ be a number of the negative signature basis elements, then

$$\boldsymbol{E}_i \widetilde{\boldsymbol{E}}_i = (-1)^r. \tag{1.24}$$

Now, what is the relation between the basis blade and its reverse? Kind of answer is that if you switch the positions of the 2 basis vectors $\boldsymbol{e}_i, \boldsymbol{e}_j$[3] included in basis blade, it results in changing signature. Either $gr(\mathbb{E}_i) = 2n+1$ or $gr(\mathbb{E}_i) = 2n$, $n \in \mathbb{N}$. When n is even, there is no signature change and if n is odd the signature will change. Function $(-1)^{\frac{(gr(\mathbb{E}_i)-1)gr(\mathbb{E}_i))}{2}}$ has precisely this property. Using (1.24) and the function from the previous sentence gives:

$$\boldsymbol{E}_i^{-1} = (-1)^r \widetilde{\boldsymbol{E}}_i = (-1)^r (-1)^{\frac{(gr(\mathbb{E}_i)-1)gr(\mathbb{E}_i)}{2}} \boldsymbol{E}_i. \tag{1.25}$$

An inverse $\boldsymbol{I}^{-1}$ to the pseudoscalar $\boldsymbol{I} = \overline{\mathbb{G}}_{p,q}[2^{p+q}]$ is then,

$$\boldsymbol{I}^{-1} = (-1)^r \widetilde{\boldsymbol{I}} = (-1)^r (-1)^{\frac{(p+q-1)(p+q)}{2}} \boldsymbol{I}.$$

Example 1.1.6. In $\mathbb{G}_{5,3}$ an inverse to the pseudoscalar $\boldsymbol{I} = e_{12345678}$ is

$$\boldsymbol{I}^{-1} = (-1)^3 (-1)^{\frac{7\cdot 8}{2}} \boldsymbol{I} = (-1)(-1)^{28} \boldsymbol{I} = -\boldsymbol{I} = -e_{12345678}.$$

1.1.6. Duality

Along with capability of perform geometric transformation, which will be performed on $\mathbb{G}_{5,3}$, the advantage of using geometric algebras comes with ability to represent a geometrical object in 2 different ways. But not only that, there exist relation between these to representations. Lest start with defining what is meant by a dual element.

Definition 1.1.16. Let $\boldsymbol{U} \in \mathbb{G}_{p,q}$, then a dual multivector $\boldsymbol{U}^*$ to the multivector $\boldsymbol{U}$ is defined as

$$\boldsymbol{U}^* = \boldsymbol{U}\boldsymbol{I}^{-1}, \tag{1.26}$$

where $\boldsymbol{I}$ is pseudoscalar and $\boldsymbol{I}^{-1}$ is its inversion.

Example 1.1.7. In case of the geometric algebra $\mathbb{G}_{5,3}$, if we follow Example 1.1.6, a dual $\boldsymbol{U}^*$ to the multivector $\boldsymbol{U} \in \mathbb{G}_{5,3}$ is $\boldsymbol{U}^* = -\boldsymbol{U}\boldsymbol{I}$. When $\boldsymbol{I}^{-1} = -\boldsymbol{I}$, it results in that the dual of the dual element will not be the original element, but rather -1-multiple of that $((\boldsymbol{I}^*)^* = (\boldsymbol{I}\boldsymbol{I}^{-1})^* = 1^* = -\boldsymbol{I})$. For an example take $\boldsymbol{E}_{116} = \boldsymbol{e}_{1367} \in \mathbb{G}_{5,3}$.

$$\begin{aligned} \boldsymbol{e}^*_{1367} &= \boldsymbol{e}_{1367}(-\boldsymbol{e}_{1235678}) = \boldsymbol{e}_{2458} = \boldsymbol{E}_{141}, \\ \boldsymbol{e}^*_{2458} &= \boldsymbol{e}_{2458}(-\boldsymbol{e}_{1235678}) = -\boldsymbol{e}_{2458} = -\boldsymbol{E}_{116}. \end{aligned}$$

Note that this computation was performed by a command in the computer applications source code the presented in the Second Chapter.

To get an idea what is motivation for using multiple representations, we know that in 2D we can set a line by a function $f(u_1) = a + bu_1$, where $a, b \in \mathbb{R}$. The representation of f in the $\mathbb{R}^2$ is set of vectors $(c, a+bc) \in \mathbb{R}^2$, $a, b, c \in \mathbb{R}$. Thus we have 2 representations of the line in space of continuous functions $C(\mathbb{R})$ and in vector space $\mathbb{R}^2$. But when we are talking about representations in geometric algebra, they are in the same algebra and that's the big advantage as we will see later. Now, lets present a null spaces.

[3] It doesn't matter which $\boldsymbol{e}_i, \boldsymbol{e}_j$ are switched. The case, where the distance between switched basis vectors is 1, is trivial. In the case of $n \geq 2$ we have to do n switches to get the first element to the seconds position, and because the seconds element is one step closer to first position, we have to do $n-1$ switches to get it to the first position. Together there are $2n-1$ switches and thus it results in changing the signature.

Definition 1.1.17. Inner-product null space (IPNS) of blade $\boldsymbol{U}_{\langle k\rangle} \in \mathbb{G}_{p,q}$ is denoted $\mathbb{NI}(\boldsymbol{U}_{\langle k\rangle})$ and defined as follows;

$$\mathbb{NI}(\boldsymbol{U}_{\langle k\rangle}) = \{\boldsymbol{u} \in \mathbb{R}^{p,q} \subset \mathbb{G}_{p,q} \ : \ \boldsymbol{u} \cdot \boldsymbol{U}_{\langle k\rangle} = 0\}. \tag{1.27}$$

Definition 1.1.18. Outer-product null space (OPNS) of blade $\boldsymbol{U}_{\langle k\rangle} \in \mathbb{G}_{p,q}$ is denoted $\mathbb{NO}(\boldsymbol{U}_{\langle k\rangle})$ and defined as follows;

$$\mathbb{NO}(\boldsymbol{U}_{\langle k\rangle}) = \{\boldsymbol{u} \in \mathbb{R}^{p,q} \subset \mathbb{G}_{p,q} \ : \ \boldsymbol{u} \wedge \boldsymbol{U}_{\langle k\rangle} = 0\}. \tag{1.28}$$

Example 1.1.8. Consider geometric algebra $\mathbb{G}_3$ and consider 2-blade $\boldsymbol{u} \wedge \boldsymbol{v} \neq 0$, where $\boldsymbol{u}, \boldsymbol{v} \in \mathbb{R}^3 \subset \mathbb{G}_3$, then

$$\mathbb{NO}(\boldsymbol{u} \wedge \boldsymbol{v}) = \{\boldsymbol{w} \in \mathbb{R}^3 \subset \mathbb{G}_3 \ : \ \boldsymbol{w} \wedge \boldsymbol{u} \wedge \boldsymbol{v} = 0\} = \{c\boldsymbol{u} + d\boldsymbol{v} \ : \ c, d \in \mathbb{R}\},$$

which is a plane generated by $\boldsymbol{u}$ and $\boldsymbol{v}$. OPNS can be seen right away, but finding IPNS of $\boldsymbol{u} \wedge \boldsymbol{v}$ is a bit more complicated.

$$\begin{aligned}
\boldsymbol{u} \wedge \boldsymbol{v} &= (u_1\boldsymbol{e}_1 + u_2\boldsymbol{e}_2 + u_3\boldsymbol{e}_3) \wedge (v_1\boldsymbol{e}_1 + v_2\boldsymbol{e}_2 + v_3\boldsymbol{e}_3) = \\
&= \langle u_1v_1\boldsymbol{e}_1\boldsymbol{e}_1\rangle_2 + \langle u_1v_2\boldsymbol{e}_1\boldsymbol{e}_2\rangle_2 + \cdots + \langle u_3v_2\boldsymbol{e}_3\boldsymbol{e}_2\rangle_2 + \langle u_3v_3\boldsymbol{e}_3\boldsymbol{e}_3\rangle_2 = \\
&= (u_1v_2 - u_2v_1)\boldsymbol{e}_{12} + (u_1v_3 - u_3v_1)\boldsymbol{e}_{13} + (u_2v_3 - u_3v_2)\boldsymbol{e}_{23}.
\end{aligned}$$

Now, let $\boldsymbol{w} \in \mathbb{R}^3 = w_1\boldsymbol{e}_1 + w_2\boldsymbol{e}_2 + w_3\boldsymbol{e}_3$ such that

$$(w_1\boldsymbol{e}_1 + w_2\boldsymbol{e}_2 + w_3\boldsymbol{e}_3) \cdot ((u_1v_2 - u_2v_1)\boldsymbol{e}_{12} + (u_1v_3 - u_3v_1)\boldsymbol{e}_{13} + (u_2v_3 - u_3v_2)\boldsymbol{e}_{23}) = 0.$$

Inner product will leave only first grade elements to be non-zero,

$$\begin{aligned}
&(w_1(u_1v_2 - u_2v_1) - w_3(u_2v_3 - u_3v_2))\boldsymbol{e}_2 + (w_1(u_1v_3 - u_3v_1) + w_2(u_2v_3 - u_3v_2))\boldsymbol{e}_3 + \\
&+(-w_2(u_1v_2 - u_2v_1) - w_3(u_1v_3 - u_3v_1))\boldsymbol{e}_1 = 0.
\end{aligned}$$

There are 3 equations with 3 unknowns and solving these equations gives us a non-trivial solution

$$\begin{aligned}
w_1 &= c(u_2v_3 - u_3v_2), \\
w_2 &= c(-u_1v_3 + u_3v_1), \\
w_3 &= c(u_1v_2 - u_2v_1),
\end{aligned}$$

where $c \in \mathbb{R}$. Therefore,

$$\begin{aligned}
\mathbb{NI}(\boldsymbol{u} \wedge \boldsymbol{v}) &= \{\boldsymbol{w} \in \mathbb{R}^3 \subset \mathbb{G}_3 \ : \ \boldsymbol{w} \cdot (\boldsymbol{u} \wedge \boldsymbol{v}) = 0\} = \\
&= \{c((u_2v_3 - u_3v_2)\boldsymbol{e}_1 + (-u_1v_3 + u_3v_1)\boldsymbol{e}_2 + (u_1v_2 - u_2v_1)\boldsymbol{e}_3) : c \in \mathbb{R}\}.
\end{aligned}$$

If we follow [10], we find that IPNS and OPNS has a relation. Let $\boldsymbol{u} \in \mathbb{R}^{p,q} \subset \mathbb{G}_{p,q}$ be a vector and let $\boldsymbol{U}_{\langle k\rangle} \in \mathbb{G}_{p,q}$ be a blade with the property $\boldsymbol{u} \cdot \boldsymbol{u} \neq 0$ and $\boldsymbol{U}_{\langle k\rangle} \cdot \boldsymbol{U}_{\langle k\rangle} \neq 0$. Now, let $\boldsymbol{I} = \overline{\mathbb{G}}_{p,q}[2^{p+q}]$ be a pseudoscalar, then

$$(\boldsymbol{u} \wedge \boldsymbol{U}_{\langle k\rangle})^* = (\boldsymbol{u} \wedge \boldsymbol{U}_{\langle k\rangle}) \cdot \boldsymbol{I}^{-1} = \boldsymbol{u} \cdot \boldsymbol{U}^*_{\langle k\rangle},$$

and therefore

$$\boldsymbol{u} \wedge \boldsymbol{U}_{\langle k \rangle} = 0 \quad \iff \quad \boldsymbol{u} \cdot \boldsymbol{U}^*_{\langle k \rangle} = 0,$$

and

$$\mathbb{NO}(\boldsymbol{U}_{\langle k \rangle}) = \mathbb{NI}(\boldsymbol{U}^*_{\langle k \rangle}). \tag{1.29}$$

In other words, an IPNS of a blade $\boldsymbol{U}_{\langle k \rangle}$ is equal to OPNS of the $\boldsymbol{U}^*_{\langle k \rangle}$. It works also in other direction, OPNS of a blade $\boldsymbol{U}_{\langle k \rangle}$ is equal to IPNS of the $\boldsymbol{U}^*_{\langle k \rangle}$. In next the Section, this property will play an important role of describing the conics.

Definition 1.1.19. A multivector $\boldsymbol{U} \in \mathbb{G}_{p,q}$ is called a null multivector, if it has following property:

$$\boldsymbol{U}\boldsymbol{U} = 0.$$

If $\mathrm{gr}(\boldsymbol{U}) = 1$, then we say that $\boldsymbol{U}$ is a null vector.

1.2. Geometric Algebra for Conics

In this Section, we introduce Geometric Algebra for Conics (GAC) together with particular embedding of two-dimensional Euclidean space. We follow [7] with the definitions, and we present examples computed and drawn by a software developed for this thesis. Firstly, we present GAC basis and its elements. Then we continue with the inner and outer product representation of some conics (for our purposes we don't have to cover all of them). We also introduce Euclidean transformations, which can be performed on both representations. This section is finished by Conic fitting algorithm (introduced by [6]), which assigns closest conic to a set of points. This algorithm is implemented and can be tested directly in the software.

1.2.1. Definition of GAC

GAC is geometric algebra $\mathbb{G}_{5,3}$. Dimension of this algebra is $2^8 = 256$, so working with large multivectors can be very demanding. Subsection 1.1.1 assumes the bilinear form in certain way and all the geometric algebra properties comes from that bilinear form. Question is what happens to the properties if we get different vector space basis. Lets write down standard basis $\overline{\mathbb{R}}^{5,3}$ of the vector space $\mathbb{R}^{5,3}$ equipped with bilinear form (1.2) represented by matrix $\overline{\boldsymbol{B}}$

$$\begin{aligned} \overline{\mathbb{R}}^{5,3} &= \begin{pmatrix} \boldsymbol{e}_1 & \boldsymbol{e}_2 & \boldsymbol{e}_3 & \boldsymbol{e}_4 & \boldsymbol{e}_5 & \boldsymbol{e}_6 & \boldsymbol{e}_7 & \boldsymbol{e}_8 \end{pmatrix}, \\ \overline{\boldsymbol{B}} &= \begin{pmatrix} 1_{5\times 5} & 0 \\ 0 & -1_{3\times 3} \end{pmatrix}, \end{aligned} \tag{1.30}$$

where $1_{n\times n}$ denotes diagonal $n \times n$ identity matrix. Due to conformal embedding of a point defined bellow we define basis $\overline{\mathbb{R}}^{5,3^*}$ which corresponding bilinear form on its basis in a matrix form vectors becomes $\boldsymbol{B}$

$$\begin{aligned}\overline{\mathbb{R}}^{5,3^*} &= \begin{pmatrix}\overline{\boldsymbol{n}}_\times & \overline{\boldsymbol{n}}_- & \overline{\boldsymbol{n}}_+ & \boldsymbol{e}_1 & \boldsymbol{e}_2 & \boldsymbol{n}_+ & \boldsymbol{n}_- & \boldsymbol{n}_\times\end{pmatrix},\\ \boldsymbol{B} &= \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1\\ 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0\\ 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0\\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0\\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0\\ 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0\\ 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0\\ -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0\end{pmatrix}.\end{aligned} \tag{1.31}$$

Let $\boldsymbol{C}$ be transition matrix between these two basis, then

$$\begin{aligned}\overline{\mathbb{R}}^{5,3^*} &= \boldsymbol{C}\overline{\mathbb{R}}^{5,3},\\ \boldsymbol{C} &= \begin{pmatrix} -\frac{1}{2} & 0 & 0 & 0 & 0 & 0 & 0 & 1\\ 0 & -\frac{1}{2} & 0 & 0 & 0 & 0 & 1 & 0\\ 0 & 0 & -\frac{1}{2} & 0 & 0 & 1 & 0 & 0\\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0\\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0\\ 0 & 0 & \frac{1}{2} & 0 & 0 & 1 & 0 & 0\\ 0 & \frac{1}{2} & 0 & 0 & 0 & 0 & 1 & 0\\ \frac{1}{2} & 0 & 0 & 0 & 0 & 0 & 0 & 1\end{pmatrix}.\end{aligned} \tag{1.32}$$

Basis $\overline{\mathbb{R}}^{5,3^*}$ then can be written in the terms of basis $\overline{\mathbb{R}}^{5,3}$, where

$$\begin{aligned} \overline{\boldsymbol{n}}_\times &= \tfrac{1}{2}(-\boldsymbol{e}_1 + \boldsymbol{e}_8), & \boldsymbol{n}_\times &= \boldsymbol{e}_1 + \boldsymbol{e}_8,\\ \overline{\boldsymbol{n}}_- &= \tfrac{1}{2}(-\boldsymbol{e}_2 + \boldsymbol{e}_7), & \boldsymbol{n}_- &= \boldsymbol{e}_2 + \boldsymbol{e}_7,\\ \overline{\boldsymbol{n}}_+ &= \tfrac{1}{2}(-\boldsymbol{e}_3 + \boldsymbol{e}_6), & \boldsymbol{n}_+ &= \boldsymbol{e}_3 + \boldsymbol{e}_6,\\ \boldsymbol{e}_1 &= \boldsymbol{e}_4 & \boldsymbol{e}_2 &= \boldsymbol{e}_5, \end{aligned} \tag{1.33}$$

Similarly we get

$$\begin{aligned}\overline{\mathbb{R}}^{5,3} &= \boldsymbol{C}^{-1}\overline{\mathbb{R}}^{5,3^*},\\ \boldsymbol{C}^{-1} &= \begin{pmatrix} -1 & 0 & 0 & 0 & 0 & 0 & 0 & 1\\ 0 & -1 & 0 & 0 & 0 & 0 & 1 & 0\\ 0 & 0 & -1 & 0 & 0 & 1 & 0 & 0\\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0\\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0\\ 0 & 0 & \frac{1}{2} & 0 & 0 & \frac{1}{2} & 0 & 0\\ 0 & \frac{1}{2} & 0 & 0 & 0 & 0 & \frac{1}{2} & 0\\ \frac{1}{2} & 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{2} \end{pmatrix}.\end{aligned}$$

Similarly $\overline{\mathbb{R}}^{5,3}$ then can be written in the terms of $\overline{\mathbb{R}}^{5,3^*}$, where

$$\begin{aligned} \boldsymbol{e}_1 &= -\overline{\boldsymbol{n}}_\times + \tfrac{1}{2}\boldsymbol{n}_\times, & \boldsymbol{e}_8 &= \overline{\boldsymbol{n}}_\times + \tfrac{1}{2}\boldsymbol{n}_\times,\\ \boldsymbol{e}_2 &= -\overline{\boldsymbol{n}}_- + \tfrac{1}{2}\boldsymbol{n}_-, & \boldsymbol{e}_7 &= \overline{\boldsymbol{n}}_- + \tfrac{1}{2}\boldsymbol{n}_-,\\ \boldsymbol{e}_3 &= -\overline{\boldsymbol{n}}_+ + \tfrac{1}{2}\boldsymbol{n}_+, & \boldsymbol{e}_6 &= \overline{\boldsymbol{n}}_+ + \tfrac{1}{2}\boldsymbol{n}_+,\\ \boldsymbol{e}_4 &= \boldsymbol{e}_1, & \boldsymbol{e}_5 &= \boldsymbol{e}_2. \end{aligned} \tag{1.34}$$

This transition comes handy in the second Chapter as it is easier to work with (1.30). To add some geometric meaning to GAC, we have to describe the embedding of $\mathbb{R}^2$ plane into the space $\mathbb{R}^{5,3}$. As we can see this transformed basis $\overline{\mathbb{R}}^{5,3^*}$ has 6 null vectors and 2 positive signature vectors.

Vectors $\boldsymbol{e}_1$ and $\boldsymbol{e}_2$ are just coordinates from $\mathbb{R}^2$. The null vectors $\overline{\boldsymbol{n}}$ and $\boldsymbol{n}$ are representing origin and infinity respectively. In terms of this basis, a point of the plane represented by vector $\boldsymbol{u} = u_1\boldsymbol{e}_1 + u_2\boldsymbol{e}_2 \in \mathbb{R}^2$ is embedded by function $\mathcal{C} : \mathbb{R}^2 \to \mathbb{G}_{5,3}$ (defined in [7]),

$$\mathcal{C}(\boldsymbol{u}) = \overline{\boldsymbol{n}}_+ + u_1\boldsymbol{e}_1 + u_2\boldsymbol{e}_2 + \frac{1}{2}(u_1^2+u_2^2)\boldsymbol{n}_+ + \frac{1}{2}(u_1^2-u_2^2)\boldsymbol{n}_- + u_1u_2\boldsymbol{n}_\times. \tag{1.35}$$

The pseudoscalar $\boldsymbol{I}$ of GAC, is of the form

$$\boldsymbol{I} = \overline{\boldsymbol{n}}_\times\overline{\boldsymbol{n}}_-\overline{\boldsymbol{n}}_+\boldsymbol{e}_1\boldsymbol{e}_2\boldsymbol{n}_+\boldsymbol{n}_-\boldsymbol{n}_\times. \tag{1.36}$$

The image $\mathcal{C}$ of the plane in $\mathbb{R}^{5,3}$ is an analogue of the conformal cone[4] and for $\boldsymbol{u} \in \mathbb{R}^2$: $\mathcal{C}(\boldsymbol{u}) \cdot \mathcal{C}(\boldsymbol{u}) = 0$. Let us calculate the inner product of two embedded points $\boldsymbol{u}, \boldsymbol{v} \in \mathbb{R}^2$:

$$\begin{aligned} \mathcal{C}(\boldsymbol{u}) \cdot \mathcal{C}(\boldsymbol{v}) &= (\overline{\boldsymbol{n}}_+ + u_1\boldsymbol{e}_1 + u_2\boldsymbol{e}_2 + \tfrac{1}{2}(u_1^2+u_2^2)\boldsymbol{n}_+ + \tfrac{1}{2}(u_1^2-u_2^2)\boldsymbol{n}_- + u_1u_2\boldsymbol{n}_\times)\cdot\\ &\quad \cdot(\overline{\boldsymbol{n}}_+ + v_1\boldsymbol{e}_1 + v_2\boldsymbol{e}_2 + \tfrac{1}{2}(v_1^2+v_2^2)\boldsymbol{n}_+ + \tfrac{1}{2}(v_1^2-v_2^2)\boldsymbol{n}_- + v_1v_2\boldsymbol{n}_\times)\\ &= u_1v_1\boldsymbol{e}_1\cdot\boldsymbol{e}_1 + u_2v_2\boldsymbol{e}_2\cdot\boldsymbol{e}_2 + \tfrac{1}{2}(v_1^2+v_2^2)\overline{\boldsymbol{n}}_+\cdot\boldsymbol{n}_+ + \tfrac{1}{2}(u_1^2+u_2^2)\boldsymbol{n}_+\cdot\overline{\boldsymbol{n}}_+ =\\ &= u_1v_1 + u_2v_2 - \tfrac{1}{2}(v_1^2+v_2^2) - \tfrac{1}{2}(u_1^2+u_2^2) =\\ &= -\tfrac{1}{2}(v_1^2 - 2v_1u_1 + u_1^2) - \tfrac{1}{2}(v_2^2 - 2v_2u_2 + u_2^2) = -\tfrac{1}{2}((v_1-u_1)^2 + (v_2-u_2)^2). \end{aligned}$$

[4] In [9], the null cone in GAC context is 4-dimensional null cone where the embedded points are placed.

If the result is multiplied by -2, then the square root of the result corresponds to the Euclidean norm of the vector $(\boldsymbol{u}-\boldsymbol{v})$, which is the distance of $\boldsymbol{u}$ and $\boldsymbol{v}$ in $||\cdot||_2$ norm. This property is common for geometric algebras with conformal embedding,

$$||\boldsymbol{u}-\boldsymbol{v}||_2^2 = -2(\mathcal{C}(\boldsymbol{u})\cdot\mathcal{C}(\boldsymbol{v})). \tag{1.37}$$

1.2.2. Inner Product Representation

In 1.27 we computed the IPNS and OPNS representation of an object A given by certain blade. Now we define IPNS of an unknown A as embedded point $\mathcal{C}(\boldsymbol{u})$ given by general vector $\boldsymbol{u}\in\mathbb{R}^2$ and we want to know what is the object A in each of the representations (A_I, A_O). Let $\boldsymbol{x}\in\mathbb{R}^{5,3}$ be a general vector[5] in terms of basis $\overline{\mathbb{R}}^{5,3^*}$, we follow [7]

$$\boldsymbol{v} = \overline{v}_\times\overline{\boldsymbol{n}}_\times + \overline{v}_+\overline{\boldsymbol{n}}_+ + \overline{v}_-\overline{\boldsymbol{n}}_- + v_1\boldsymbol{e}_1 + v_2\boldsymbol{e}_2 + v_+\boldsymbol{n}_+ + v_-\boldsymbol{n}_- + v_\times\boldsymbol{n}_\times.$$

Now its inner product with the embedded point is:

$$\begin{aligned}
\mathcal{C}(\boldsymbol{u})\cdot\boldsymbol{v} &= v_+\overline{\boldsymbol{n}}_+\cdot\boldsymbol{n}_+ + u_1v_1\boldsymbol{e}_1\cdot\boldsymbol{e}_1 + u_2v_2\boldsymbol{e}_2\cdot\boldsymbol{e}_2 + \tfrac{1}{2}(u_1^2+u_2^2)\overline{v}_+\boldsymbol{n}_+\cdot\overline{\boldsymbol{n}}_+ \\
&+ \tfrac{1}{2}(u_1^2-u_2^2)\overline{v}_-\boldsymbol{n}_-\cdot\overline{\boldsymbol{n}}_- + u_1u_2\overline{v}_\times\boldsymbol{n}_\times\cdot\overline{\boldsymbol{n}}_\times = \\
&= -v_+ + u_1v_1 + u_2v_2 - \tfrac{1}{2}(u_1^2+u_2^2)\overline{v}_+ - \tfrac{1}{2}(u_1^2-u_2^2)\overline{v}_- - u_1u_2\overline{v}_\times = \\
&= -v_+ + v_1u_1 + v_2u_2 - \tfrac{1}{2}(\overline{v}_+ + \overline{v}_-)u_1^2 - \tfrac{1}{2}(\overline{v}_+ - \overline{v}_-)u_2^2 - u_1u_2\overline{v}_\times.
\end{aligned} \tag{1.38}$$

It is a general polynomial of degree two, thus this inner product shows that IPNS of embedded point is the most general equation for conics;

$$Au_1^2 + 2Bu_1u_2 + Cu_2^2 + 2Du_1 + 2Eu_2 + F = 0. \tag{1.39}$$

We see that the term $v_-\boldsymbol{n}_- + v_\times\boldsymbol{n}_\times$ is orthogonal to the all embedded points. So the inner representation of a conic can be defined as a vector

$$Q_I = \overline{v}_\times\overline{\boldsymbol{n}}_\times + \overline{v}_+\overline{\boldsymbol{n}}_+ + \overline{v}_-\overline{\boldsymbol{n}}_- + v_1\boldsymbol{e}_1 + v_2\boldsymbol{e}_2 + v_+\boldsymbol{n}_+. \tag{1.40}$$

Conic section can also be read off its matrix representation with coefficients from (1.39) and also from last term in (1.38),

$$Q = \begin{pmatrix} A & B & D \\ B & C & E \\ D & E & F \end{pmatrix} = \begin{pmatrix} -\frac{1}{2}(\overline{v}_+ + \overline{v}_-) & -\frac{1}{2}\overline{v}_\times & \frac{1}{2}v_1 \\ -\frac{1}{2}\overline{v}_\times & -\frac{1}{2}(\overline{v}_+ - \overline{v}_-) & \frac{1}{2}v_2 \\ \frac{1}{2}v_1 & \frac{1}{2}v_2 & -v_+ \end{pmatrix}. \tag{1.41}$$

As we can see from (1.38), the matrix can be computed also using the inner product and Q_I;

$$Q = \begin{pmatrix} Q_I\cdot\frac{1}{2}(\boldsymbol{n}_+ + \boldsymbol{n}_-) & Q_I\cdot\frac{1}{2}\boldsymbol{n}_\times & Q_I\cdot\frac{1}{2}\boldsymbol{e}_1 \\ Q_I\cdot\frac{1}{2}\boldsymbol{n}_\times & Q_I\cdot\frac{1}{2}(\boldsymbol{n}_+ - \boldsymbol{n}_-) & Q_I\cdot\frac{1}{2}\boldsymbol{e}_2 \\ Q_I\cdot\frac{1}{2}\boldsymbol{e}_1 & Q_I\cdot\frac{1}{2}\boldsymbol{e}_2 & Q_I\cdot\overline{\boldsymbol{n}}_+ \end{pmatrix}. \tag{1.42}$$

[5] Remember that IPNS is set of vectors from $\mathbb{R}^{p,q}\subset\mathbb{G}_{p,q}$ according to term 1.27.

Example 1.2.1. There are three types of non-degenerate conics. First are ellipses, the simplest ellipse is axes-aligned and centered in the origin (i.e. $A = b^2$, $C = a^2$, $F = -a^2b^2$). In this case the equation (1.39) becomes:

$$b^2u_1^2 + a^2u_2^2 - a^2b^2 = 0. \tag{1.43}$$

Now, according to the last term in (1.38),

$$\tfrac{1}{2}(\overline{v}_+ + \overline{v}_-)u_1^2 + \tfrac{1}{2}(\overline{v}_+ - \overline{v}_-)u_2^2 + v_+ = 0.$$

From equation (1.43) we see that, $v_+ = -a^2b^2$ and $\overline{v}_+ + \overline{v}_- = 2b^2$, resp. $\overline{v}_+ - \overline{v}_- = 2a^2$. Solving this equations give the GAC vector for above described ellipse

$$E_I = (a^2 + b^2)\overline{\boldsymbol{n}}_+ + (-a^2 + b^2)\overline{\boldsymbol{n}}_- - a^2b^2\boldsymbol{n}_+. \tag{1.44}$$

The general equation's coefficients can be obtained from known semi-major axis a, semi-minor axis b, center coordinates $(c_1, c_2) \in \mathbb{R}^2$, and rotation angle θ using the formulae:

$$\begin{aligned}
A &= -\tfrac{1}{2}(\overline{v}_+ + \overline{v}_-) = a^2\sin^2\theta + b^2\cos^2\theta,\\
2B &= -\overline{v}_\times = 2(b^2 - a^2)\sin\theta\cos\theta,\\
C &= -\tfrac{1}{2}(\overline{v}_+ - \overline{v}_-) = a^2\cos^2\theta + b^2\sin^2\theta,\\
2D &= v_1 = -2Ac_1 - 2Bc_2,\\
2E &= v_2 = -2Bc_1 - 2Cc_2,\\
F &= -v_+ = Ac_1^2 + 2Bc_1c_2 + Cc_2^2 - a^2b^2.
\end{aligned}$$

We can see that $\overline{v}_\times, x_1, x_2, -x_+$ right away. But some of these depends on A, C. Now let us find the values of $\overline{v}_+, \overline{v}_-$:

$$\begin{aligned}
\overline{v}_+ &= -(A + C) = -a^2(\sin^2\theta + \cos^2\theta) - b^2(\sin^2\theta + \cos^2\theta) = -(a^2 + b^2),\\
\overline{v}_- &= -(A - C) = a^2(-\sin^2\theta + \cos^2\theta) - b^2(\cos^2\theta - \sin^2\theta) =\\
&= a^2\cos 2\theta - b^2\cos 2\theta = (a^2 - b^2)\cos 2\theta,\\
\overline{v}_\times &= -2(b^2 - a^2)\sin\theta\cos\theta = (a^2 - b^2)\sin 2\theta,\\
v_1 &= -2Ac_1 - Bc_2 = (\overline{v}_+ + \overline{v}_-)c_1 - \overline{v}_\times c_2 =\\
&= (-(a^2 + b^2) - (a^2 - b^2)\cos 2\theta)c_1 + ((a^2 - b^2)\sin 2\theta)c_2,\\
v_2 &= -Bc_1 - 2Cc_2 = -\overline{v}_\times c_1 + (\overline{v}_+ - \overline{v}_-)c_2 =\\
&= ((a^2 - b^2)\sin 2\theta)c_1 + (-(a^2 + b^2) + (a^2 - b^2)\cos 2\theta)c_2,\\
v_+ &= \tfrac{1}{2}(-2Ac_1^2 - 2Bc_1c_2 - 2Cc_2^2 + 2a^2b^2) =\\
&= \tfrac{1}{2}((\overline{v}_+ + \overline{v}_-)c_1^2 - 2\overline{v}_\times c_1c_2 + (\overline{v}_+ - \overline{v}_-)c_2^2 + 2a^2b^2) =\\
&= \tfrac{1}{2}((-(a^2 + b^2) - (a^2 - b^2)\cos 2\theta)c_1^2 + (a^2 - b^2)\sin 2\theta c_1c_2 +\\
&\quad +(-(a^2 + b^2) + (a^2 - b^2)\cos 2\theta)c_2^2 + 2a^2b^2).
\end{aligned}$$

Now, to simplify some of these terms, let the whole equation be multiplied by $-\frac{1}{a^2+b^2}$;

$$\begin{aligned}
\overline{v}_+ &= \gamma, \\
\overline{v}_- &= -\alpha\cos 2\theta, \\
\overline{v}_\times &= -\alpha\sin 2\theta, \\
v_1 &= (\gamma-\alpha\cos 2\theta)c_1 - (\alpha\sin 2\theta)c_2, \\
v_2 &= -\alpha\sin 2\theta c_1 + (\gamma+\alpha\cos 2\theta)c_2, \\
v_+ &= \tfrac{1}{2}((\gamma-\alpha\cos 2\theta)c_1^2 - 2\alpha\sin 2\theta c_1c_2 + (\gamma+\alpha\cos 2\theta)c_2^2 - \beta),
\end{aligned}$$

where $\alpha = a^2-b^2, \beta = 2a^2b^2, \gamma = a^2+b^2$. The general vector for the ellipse in GAC[6] becomes

$$\begin{aligned}
E_I \;=\; & -(\alpha\sin 2\theta)\overline{\boldsymbol{n}}_\times + \gamma\overline{\boldsymbol{n}}_+ - (\alpha\cos 2\theta)\overline{\boldsymbol{n}}_- + \\
& +((\gamma-\alpha\cos 2\theta)c_1 - (\alpha\sin 2\theta)c_2)\boldsymbol{e}_1 + (-\alpha\sin 2\theta c_1 + (\gamma+\alpha\cos 2\theta)c_2)\boldsymbol{e}_2 + \\
& +\tfrac{1}{2}((\gamma-\alpha\cos 2\theta)c_1^2 - 2\alpha\sin 2\theta c_1c_2 + (\gamma+\alpha\cos 2\theta)c_2^2 - \beta)\boldsymbol{n}_+ ,
\end{aligned} \tag{1.45}$$

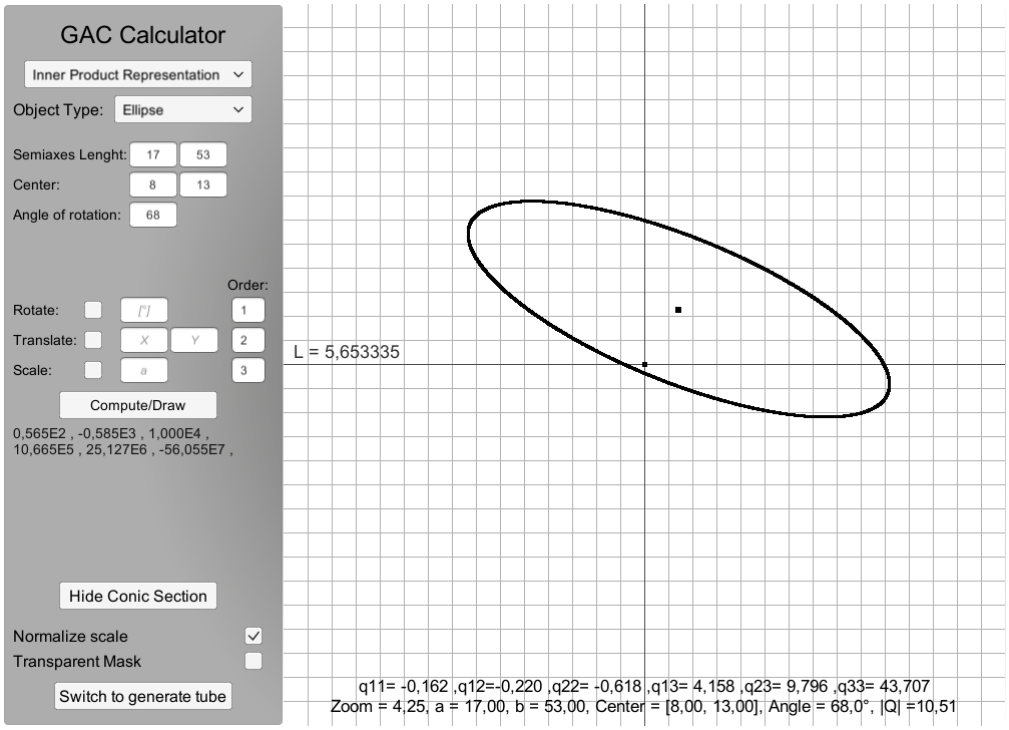


Figure 1.1: Ellipse given from IPNS representation

[6]We got the same result as [7].

In Figure 1.1 we can see an ellipse given by (1.45). Number L denotes the size of one square. The ellipse's parameters are: $a = 17, b = 53, c_1 = 8, c_2 = 13, \theta = 68°$. Resulting coefficients are

$$Q = \begin{pmatrix} -0.16174 & -0.22029 & 4.15767 \\ -0.22029 & -0.61797 & 9.79595 \\ 4.15767 & 9.79595 & 43.70689 \end{pmatrix}.$$

Note that, for esthetic reasons, the result is normalized: $Q = \frac{1}{|A|+|B|+|C|}Q'$.

Example 1.2.2. The next example is a special case of the ellipse (1.45), when $a = b = r$ and $\theta = 0°$, we get equation for a circle with radius r and center in $(c_1, c_2) \in \mathbb{R}^2$;

$$C_I \quad = \quad \overline{\boldsymbol{n}}_+ + c_1\boldsymbol{e}_1 + c_2\boldsymbol{e}_2 + \tfrac{1}{2}(c_1^2 + c_2^2 - r^2)\boldsymbol{n}_+ \,. \tag{1.46}$$

Example 1.2.3. The next example is hyperbola, which can be obtained in similar way like the ellipse in Example 1.2.1. In GAC, an axes-aligned hyperbola centered in the origin with semi-major axis a, semi-minor axis b is obtained as;

$$H_I = (a^2 + b^2)\overline{\boldsymbol{n}}_+ + (-a^2 + b^2)\overline{\boldsymbol{n}}_- + a^2b^2\boldsymbol{n}_+. \tag{1.47}$$

Hyperbola with semi-major axis a, semi-minor axis b, center coordinates $(c_1, c_2) \in \mathbb{R}^2$, and rotation angle θ is defined by the same equation (1.45);

$$\begin{aligned} H_I \quad = \quad & -(\alpha \sin 2\theta)\overline{\boldsymbol{n}}_\times + \gamma\overline{\boldsymbol{n}}_+ - (\alpha\cos 2\theta)\overline{\boldsymbol{n}}_- + \\ & +((\gamma - \alpha\cos 2\theta)c_1 - (\alpha \sin 2\theta)c_2)\boldsymbol{e}_1 + (-\alpha \sin 2\theta c_1 + (\gamma + \alpha\cos 2\theta)c_2)\boldsymbol{e}_2 + \\ & +\tfrac{1}{2}((\gamma - \alpha\cos 2\theta)c_1^2 - 2\alpha \sin 2\theta c_1c_2 + (\gamma + \alpha\cos 2\theta)c_2^2 - \beta)\boldsymbol{n}_+ \,, \end{aligned} \tag{1.48}$$

but $\alpha = a^2 + b^2, \beta = -2a^2b^2, \gamma = a^2 - b^2$. Proposed and proved in [7].

In Figure 1.2 we can see a hyperbola given by (1.48). The hyperbola's parameters are: $a = 54, b = 22, c_1 = -34, c_2 = -23, \theta = -12°$. Resulting coefficients are

$$Q = \begin{pmatrix} 0.08875 & -0.18208 & -1.17037 \\ -0.18208 & -0.72917 & -22.96165 \\ -1.17037 & -22.96165 & -939.56047 \end{pmatrix}.$$

1.2.3. Outer Product Representation

Recall that due to duality shown in (1.29), we can compute outer product representation of an object A by finding dual to A_I. But this dual A_I^* is always a multivector of the form $A_O \wedge \overline{\boldsymbol{n}}_- \wedge \overline{\boldsymbol{n}}_\times$. It follows from the equation (1.40) of a general conic section that the basis vectors $\boldsymbol{n}_-, \boldsymbol{n}_\times$ are orthogonal to the general conic Q_I. According to [7], we will call the term A_O as the outer representation of an entity A if and only if

$$A = \{\boldsymbol{u} \in \mathbb{R}^2 : \mathcal{C}(\boldsymbol{u}) \wedge A_O \wedge \overline{\boldsymbol{n}}_- \wedge \overline{\boldsymbol{n}}_\times = 0\}.$$

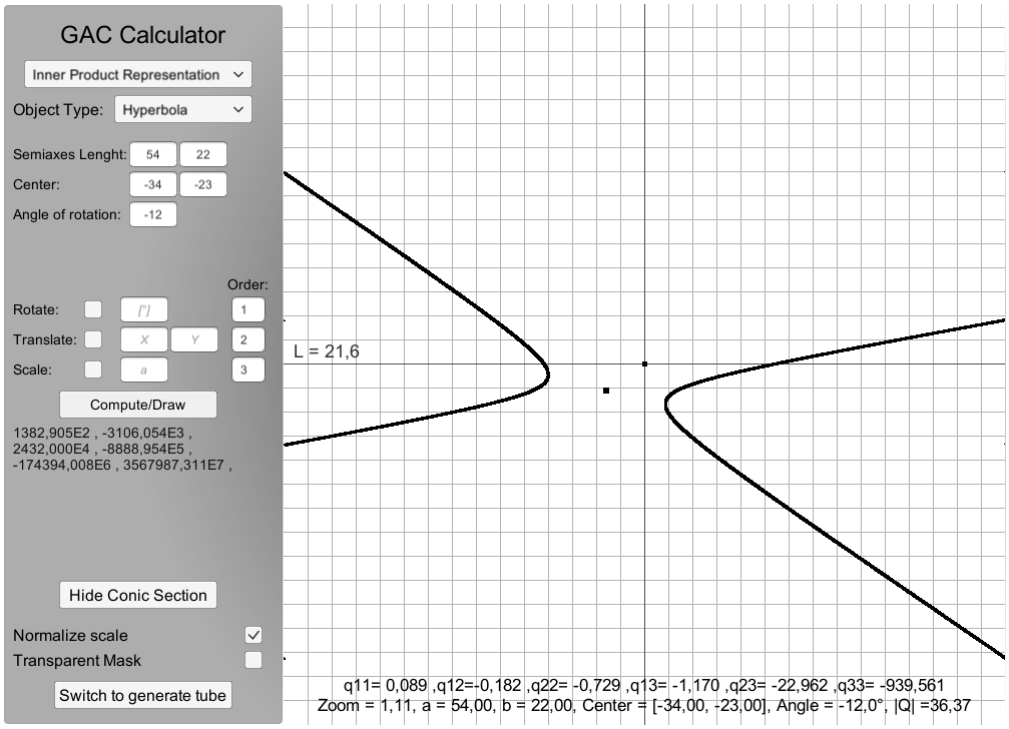


Figure 1.2: Hyperbola given from IP representation

To preserve the duality between A_I and A_O, we define duality between them as

$$\begin{aligned} A_O &= (A_I \wedge \boldsymbol{n}_- \wedge \boldsymbol{n}_\times)^*, \\ A_I &= (A_O \wedge \overline{\boldsymbol{n}}_- \wedge \overline{\boldsymbol{n}}_\times)^*. \end{aligned} \tag{1.49}$$

The outer representation of structures, in geometric algebra CGA in [9], were minimal number of points that can define the structure. For example sphere in CGA is the outer product of points lying on it. In GAC, the structures are obtained similarly.

Example 1.2.4. The outer product representation of a general conic Q in GAC is given by five point lying on it. Let $\boldsymbol{u}_1, \ldots, \boldsymbol{u}_5 \in \mathbb{R}^2$, then

$$Q_O = \mathcal{C}(\boldsymbol{u}_1) \wedge \mathcal{C}(\boldsymbol{u}_2) \wedge \mathcal{C}(\boldsymbol{u}_3) \wedge \mathcal{C}(\boldsymbol{u}_4) \wedge \mathcal{C}(\boldsymbol{u}_5). \tag{1.50}$$

In Figure 1.3 we can see an ellipse given by (1.50). The ellipse is spanned by points $(-4.9, -6.2), (-3.4, -4.2), (6, -10), (5.6, 1), (11.9, -2.3)$. Results coefficients are

$$Q = \begin{pmatrix} -0.29966 & 0.20190 & 2.04867 \\ 0.20190 & -0.49844 & -3.43575 \\ 2.04867 & -3.43575 & -8.43918 \end{pmatrix}.$$

Changing the point $(-4.9, -6.2) \to (-8, -6.2)$ results in a hyperbola showed in Figure 1.4 with coefficients

$$Q = \begin{pmatrix} -0.27045 & 0.41805 & 2.04867 \\ 0.41805 & -0.31150 & 2.64056 \\ 2.04867 & 2.64056 & -17.75187 \end{pmatrix}.$$

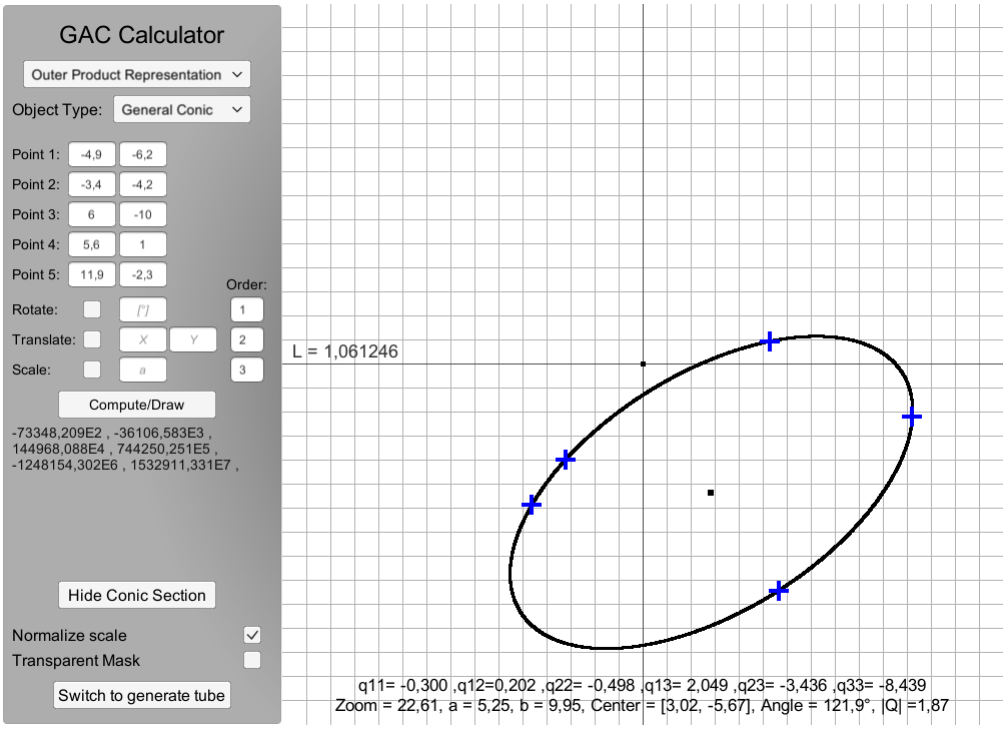


Figure 1.3: Ellipse spanned by 5 points

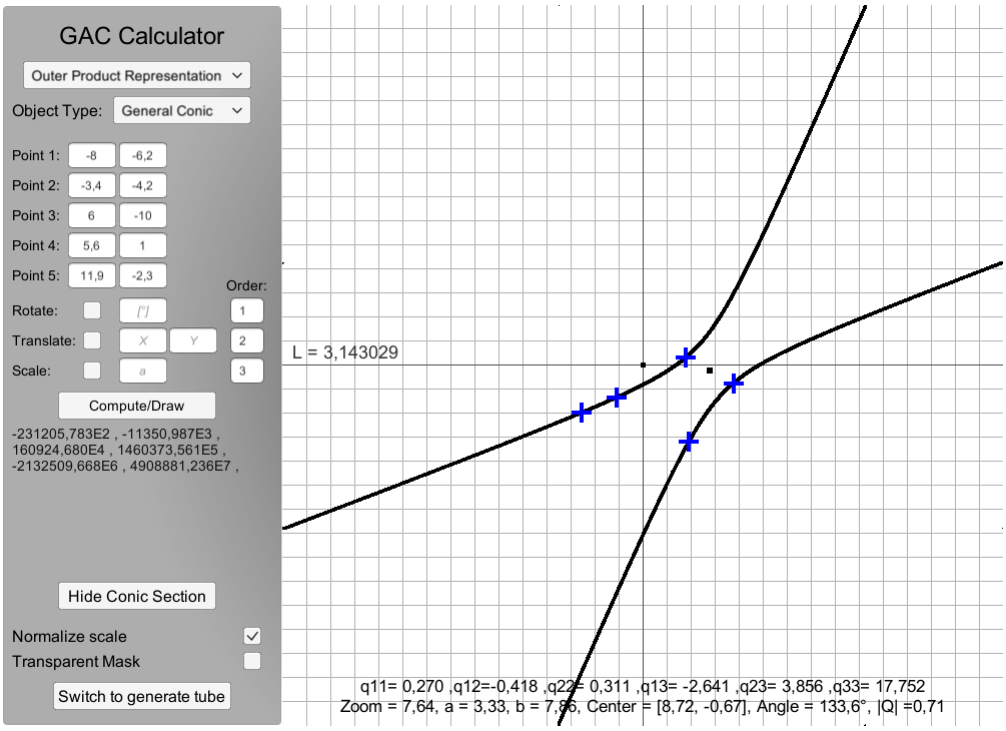


Figure 1.4: Hyperbola spanned by 5 points

Example 1.2.5. An axes-aligned conic Q_O^{al} in GAC is given by four points lying on it. Let $\boldsymbol{u}_1, \ldots, \boldsymbol{u}_4 \in \mathbb{R}^2$, then

$$Q_O^{al} = \mathcal{C}(\boldsymbol{u}_1) \wedge \mathcal{C}(\boldsymbol{u}_2) \wedge \mathcal{C}(\boldsymbol{u}_3) \wedge \mathcal{C}(\boldsymbol{u}_4) \wedge \boldsymbol{n}_\times. \tag{1.51}$$

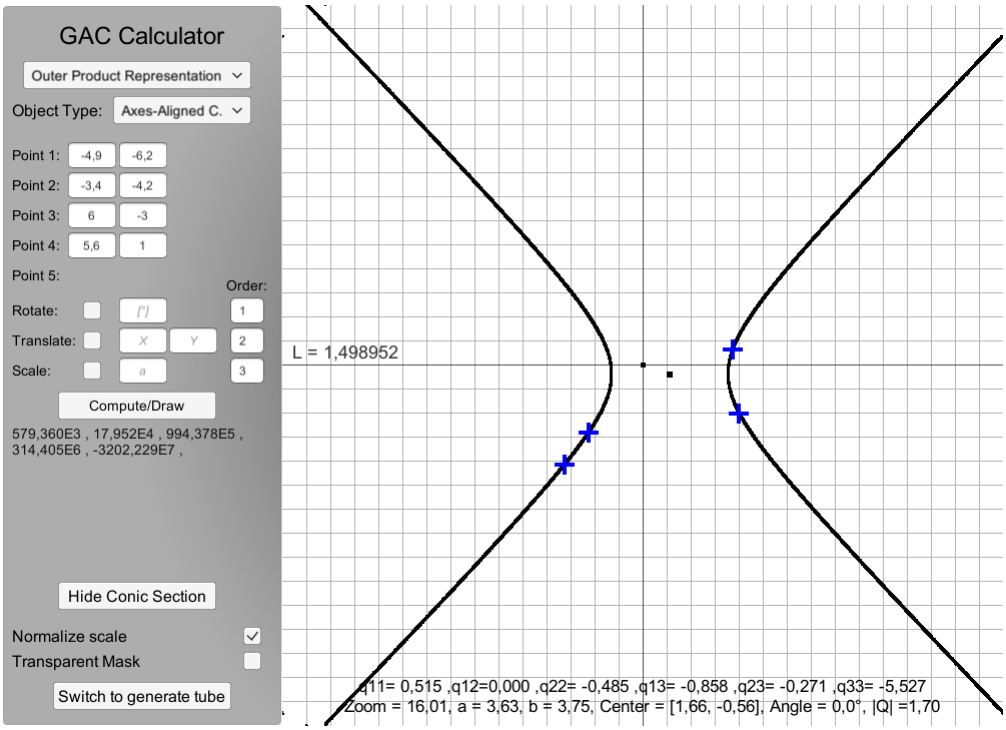


Figure 1.5: Axes-aligned hyperbola spanned by 4 points

Figure 1.5 shows a hyperbola given by formula (1.51). The hyperbola is given by points $(-4.9, -6.2), (-3.4, -4.2), (6, -3), (5.6, 1)$. Resulting coefficients are

$$Q = \begin{pmatrix} -0.51549 & 0.00000 & 0.85817 \\ 0.00000 & 0.48451 & 0.27134 \\ 0.85817 & 0.27134 & 5.52718 \end{pmatrix}.$$

Example 1.2.6. A circle C_O in GAC is given by three points lying on it. Let $\boldsymbol{u}_1, \boldsymbol{u}_2, \boldsymbol{u}_3 \in \mathbb{R}^2$, then

$$C_O = \mathcal{C}(\boldsymbol{u}_1) \wedge \mathcal{C}(\boldsymbol{u}_2) \wedge \mathcal{C}(\boldsymbol{u}_3) \wedge \boldsymbol{n}_- \wedge \boldsymbol{n}_\times. \tag{1.52}$$

Finally, Figure 1.6 shows a circle given by formula (1.52). The circle is given by points $(0, -6.2), (-3.4, -4.2), (6, -3)$. Resulting coefficients are

$$Q = \begin{pmatrix} 0.50000 & 0.00000 & -0.46643 \\ 0.00000 & 0.50000 & 0.36206 \\ -0.46643 & 0.36206 & -14.73042 \end{pmatrix}.$$

Note that the conic parameters can be easily extracted. Subsection 2.2.3 shows how to get conic parameters from its matrix representation.

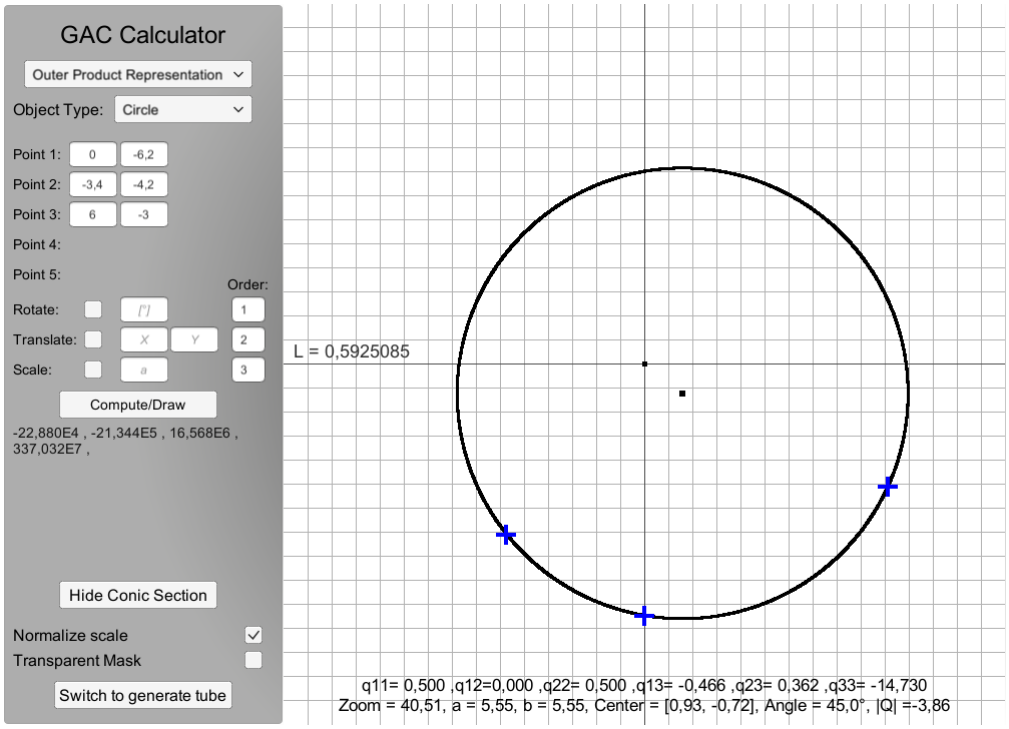


Figure 1.6: Circle spanned by 3 points

1.2.4. Euclidean Transformations and Scaling

One of the reasons why to use the geometric algebras is simple perforation of Euclidean transformations. These are realised an element called versor $\boldsymbol{R} \in \mathbb{G}_{p,q}$ and its reverse. Each of the transformation is performed on ellipse from Figure 1.3. Each of the transformations is proposed and proved in [7].

Example 1.2.7. The first transformation is rotating by angle α around the origin given by $\boldsymbol{R} = \boldsymbol{R}_+(\boldsymbol{R}_1 \wedge \boldsymbol{R}_2)$, where

$$\begin{aligned} \boldsymbol{R}_+ &= \cos(\tfrac{\alpha}{2}) + \sin(\tfrac{\alpha}{2})\boldsymbol{e}_1 \wedge \boldsymbol{e}_2, \\ \boldsymbol{R}_1 &= \cos(\alpha) + \sin(\alpha)\overline{\boldsymbol{n}}_\times \wedge \boldsymbol{n}_-, \\ \boldsymbol{R}_2 &= \cos(\alpha) - \sin(\alpha)\overline{\boldsymbol{n}}_- \wedge \boldsymbol{n}_\times. \end{aligned} \tag{1.53}$$

Coefficients for the ellipse from Figure 1.7, rotated by rotor (and its reverse) with parameter $\alpha = -143°$, are

$$Q = \begin{pmatrix} -0.21190 & -0.04760 & 0.51498 \\ -0.04760 & -0.74050 & 4.74571 \\ 0.51498 & 4.74571 & -10.07081 \end{pmatrix}.$$

Rotor in terms of basis (1.30), $\boldsymbol{R} = 0.20238 - 0.15251\boldsymbol{e}_{12} - 0.60486\boldsymbol{e}_{45} - 0.15251\boldsymbol{e}_{78} + 0.45579\boldsymbol{e}_{1245} + 0.11492\boldsymbol{e}_{1278} + 0.45579\boldsymbol{e}_{4578} - 0.34347\boldsymbol{e}_{124578}$.

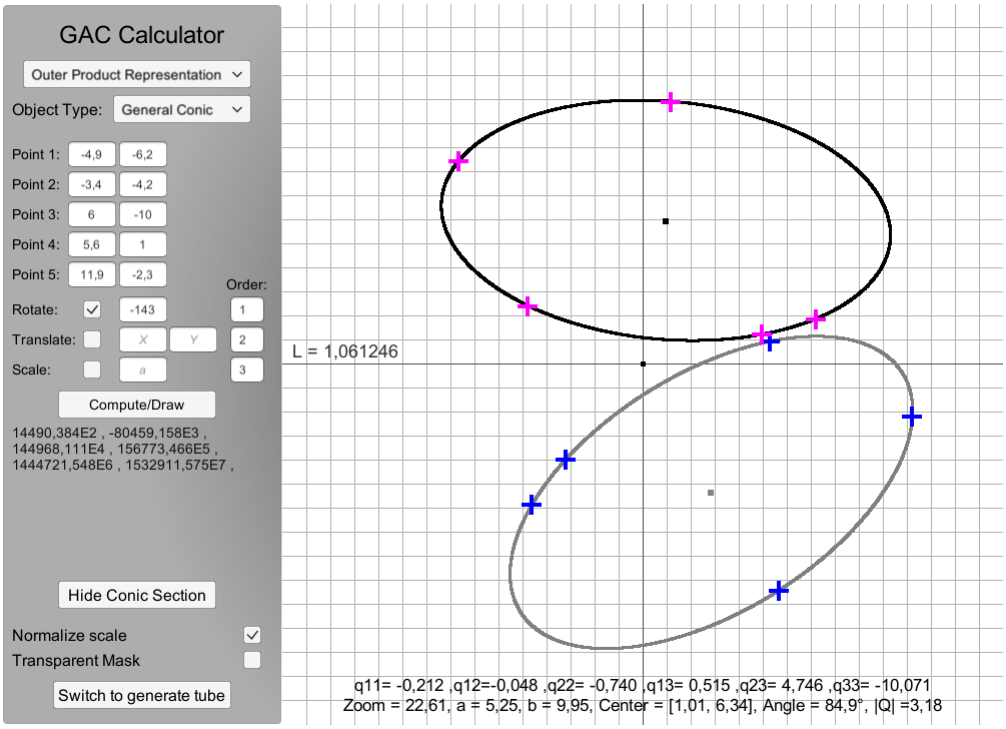


Figure 1.7: Ellipse rotated by rotor

Example 1.2.8. The next transformation is translation in direction $\boldsymbol{e}_1$ with length a given by translator $\boldsymbol{T}_{\boldsymbol{e}_1} = \boldsymbol{T}_+\boldsymbol{T}_-\boldsymbol{T}_\times$ where

$$\begin{aligned} \boldsymbol{T}_+ &= 1 - \tfrac{1}{2}a\boldsymbol{e}_1 \wedge \boldsymbol{n}_+, \\ \boldsymbol{T}_- &= 1 - \tfrac{1}{2}a\boldsymbol{e}_1 \wedge \boldsymbol{n}_- + \tfrac{1}{4}a^2\boldsymbol{n}_+ \wedge \boldsymbol{n}_-, \\ \boldsymbol{T}_\times &= 1 - \tfrac{1}{2}a\boldsymbol{e}_2 \wedge \boldsymbol{n}_\times. \end{aligned} \tag{1.54}$$

In case of translation in direction $\boldsymbol{e}_2$, we have tranlator $\boldsymbol{T}_{\boldsymbol{e}_2} = \boldsymbol{T}_+\boldsymbol{T}_-\boldsymbol{T}_\times$ where

$$\begin{aligned} \boldsymbol{T}_+ &= 1 - \tfrac{1}{2}a\boldsymbol{e}_2 \wedge \boldsymbol{n}_+, \\ \boldsymbol{T}_- &= 1 + \tfrac{1}{2}a\boldsymbol{e}_2 \wedge \boldsymbol{n}_- - \tfrac{1}{4}a^2\boldsymbol{n}_+ \wedge \boldsymbol{n}_-, \\ \boldsymbol{T}_\times &= 1 - \tfrac{1}{2}a\boldsymbol{e}_1 \wedge \boldsymbol{n}_\times. \end{aligned} \tag{1.55}$$

Coeficients for the ellipse from Figure 1.8, translated by both translators (and its reverses) with parameter $a_1 = -3.02, a_2 = 5.67$, are

$$Q = \begin{pmatrix} -0.29966 & 0.20190 & -0.00109 \\ 0.20190 & -0.49844 & 0.00014 \\ -0.00109 & 0.00014 & 17.22443 \end{pmatrix}$$

$\boldsymbol{T}_{\boldsymbol{e}_1} = 1 - 1.51\boldsymbol{e}_{15} - 1.51\boldsymbol{e}_{24} - 1.51\boldsymbol{e}_{34} + 1.51\boldsymbol{e}_{46} + 1.51\boldsymbol{e}_{47} + 1.51\boldsymbol{e}_{58} + 2.2801\boldsymbol{e}_{1245} + 2.2801\boldsymbol{e}_{1345} + 2.2801\boldsymbol{e}_{1456} + 2.2801\boldsymbol{e}_{1457} - 2.2801\boldsymbol{e}_{2458} - 2.2801\boldsymbol{e}_{3458} - 2.2801\boldsymbol{e}_{4568} - 2.2801\boldsymbol{e}_{4578}$.
$\boldsymbol{T}_{\boldsymbol{e}_2} = 1 + 2.835\boldsymbol{e}_{15} + 2.835\boldsymbol{e}_{24} + 2.835\boldsymbol{e}_{34} - 2.835\boldsymbol{e}_{46} - 2.835\boldsymbol{e}_{47} - 2.835\boldsymbol{e}_{58} + 8.03722\boldsymbol{e}_{1245} + 8.03722\boldsymbol{e}_{1345} + 8.03722\boldsymbol{e}_{1456} + 8.03722\boldsymbol{e}_{1457} - 8.03722\boldsymbol{e}_{2458} - 8.03722\boldsymbol{e}_{3458} - 8.03722\boldsymbol{e}_{4568} - 8.03722\boldsymbol{e}_{4578}$.

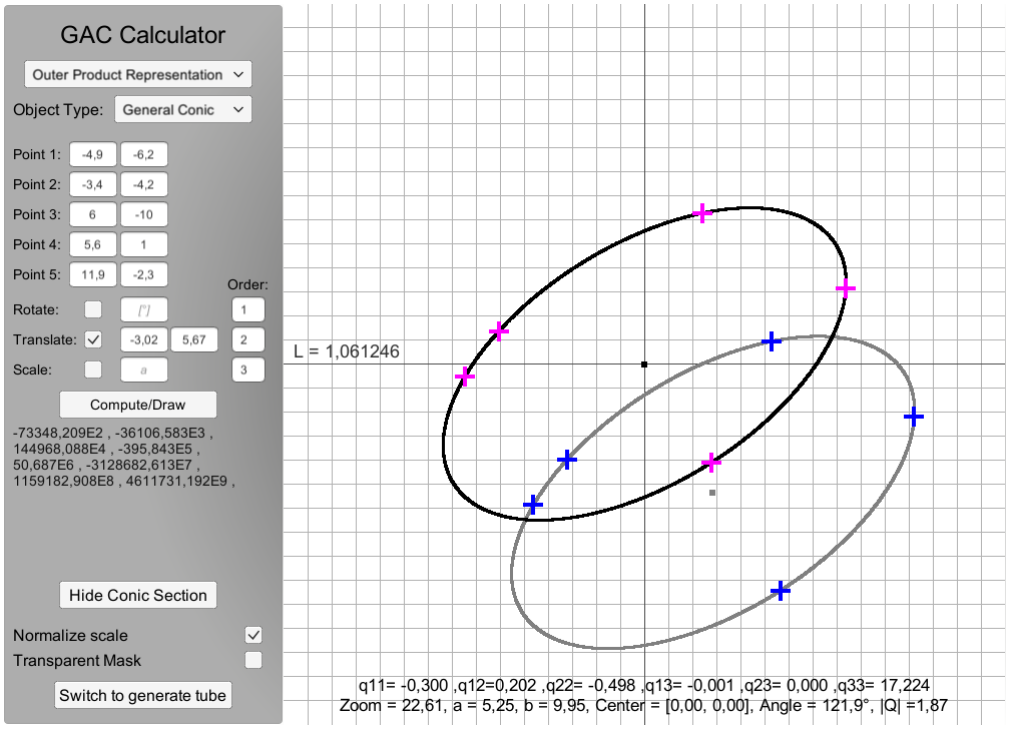


Figure 1.8: Ellipse translated by combination of translators

Example 1.2.9. Scale by α in GAC is given by scalor $\boldsymbol{S} = \boldsymbol{S}_+\boldsymbol{S}_-\boldsymbol{S}_\times$, where

$$\begin{aligned} \boldsymbol{S}_+ &= \tfrac{\alpha+1}{2\sqrt{\alpha}} + \tfrac{\alpha-1}{2\sqrt{\alpha}}\overline{\boldsymbol{n}}_+ \wedge \boldsymbol{n}_+, \\ \boldsymbol{S}_- &= \tfrac{\alpha+1}{2\sqrt{\alpha}} + \tfrac{\alpha-1}{2\sqrt{\alpha}}\overline{\boldsymbol{n}}_- \wedge \boldsymbol{n}_-, \\ \boldsymbol{S}_\times &= \tfrac{\alpha+1}{2\sqrt{\alpha}} + \tfrac{\alpha-1}{2\sqrt{\alpha}}\overline{\boldsymbol{n}}_\times \wedge \boldsymbol{n}_\times. \end{aligned} \tag{1.56}$$

Coefficients for the ellipse from Figure 1.9, scaled by scalor with parameter $\alpha = 2.3$, are

$$Q = \begin{pmatrix} -0.29966 & 0.20190 & 4.71194 \\ 0.20190 & -0.49844 & -7.90222 \\ 4.71194 & -7.90222 & -44.64327 \end{pmatrix}$$

$\boldsymbol{S} = 1.28783 - 0.50733\boldsymbol{e}_{18} - 0.50733\boldsymbol{e}_{27} - 0.50733\boldsymbol{e}_{36} + 0.19986\boldsymbol{e}_{1278} + 0.19986\boldsymbol{e}_{1368} + 0.19986\boldsymbol{e}_{2367} - 0.07873\boldsymbol{e}_{123678}$.

1.2.5. Conic Fitting

The goal of this subsection is to find a conic by interpolating a set of points. Interpolation is performed by conic fitting algorithm introduced in [6]. Basically we solve optimization problem described below. We have a general conic Q represented by vector Q_I given in (1.40), but in matrix form

$$Q_I = (\overline{v}_\times \;\; \overline{v}_- \;\; \overline{v}_- \;\; v_1 \;\; v_2 \;\; v_+ \;\; 0 \;\; 0)^T. \tag{1.57}$$

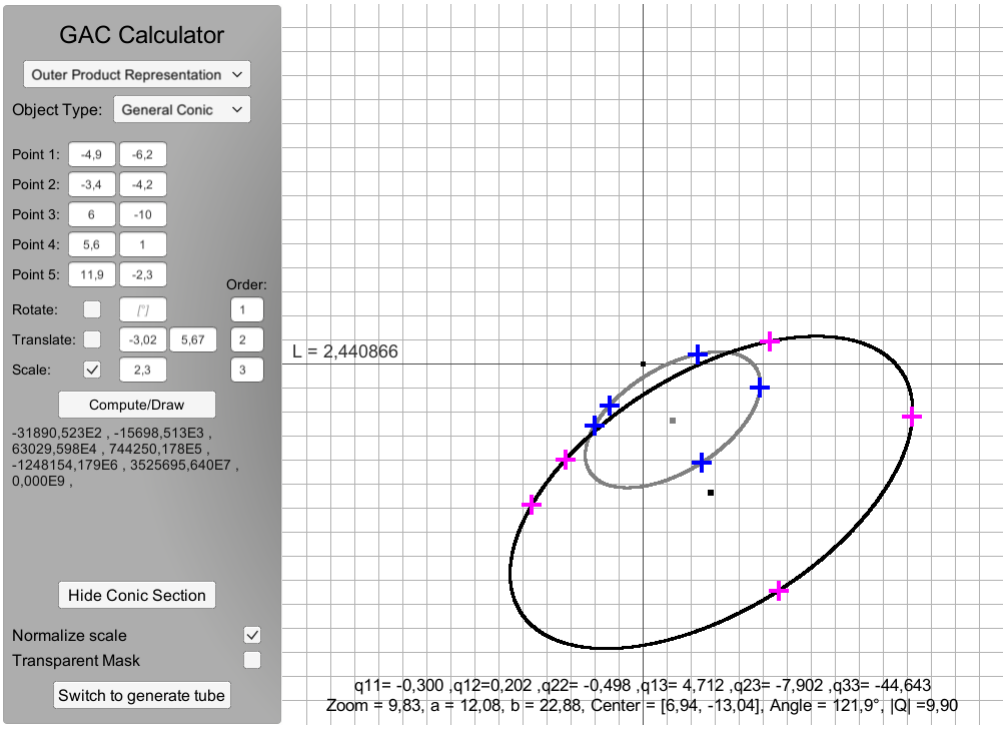


Figure 1.9: Ellipse scaled by scalor

The point in the plane represented by vector $\boldsymbol{u} = u_1\boldsymbol{e}_1 + u_2\boldsymbol{e}_2 \in \mathbb{R}^2$ and embedded to GAC defined in (1.35) will be now written in matrix form,

$$\mathcal{C}(\boldsymbol{u}) = (0 \;\; 0 \;\; 1 \;\; u_1 \;\; u_2 \;\; \tfrac{1}{2}(u_1^2 + u_2^2) \;\; \tfrac{1}{2}(u_1^2 - u_2^2) \;\; u_1u_2)^T. \tag{1.58}$$

In subsection 1.2.2 we defined conics in way that inner product $Q_I \cdot \mathcal{C}(\boldsymbol{u})$ is zero when the point lie on the conic. We have to point out that the value of $(Q_I \cdot \mathcal{C}(\boldsymbol{u}))^2$ is getting smaller as the point is closer to the conic. For a set of points $\mathbb{U}$ we define objective function (cost function) given by

$$Q \to \sum_i (\mathcal{C}(\boldsymbol{u}_i) \cdot Q)^2, \quad \boldsymbol{u}_i \in \mathbb{U}. \tag{1.59}$$

Conic closest to all points in $\mathbb{U}$ minimizes this function. As the simple solution $Q = 0$ fits every $\mathbb{U}$, we are not interested in such geometrically meaningless Q. Thus we consider the natural geometric constraint

$$QQ = 1. \tag{1.60}$$

Note this the geometric product in this case[7] is equal to inner product. Problem (1.59) with constrain (1.60) can be generalized with geometric product using bilinear form (1.30),

$$\mathcal{C}(\boldsymbol{u}_i) \cdot Q = \mathcal{C}(\boldsymbol{u}_i)\boldsymbol{B}Q,$$

where matrix multiplication is executed using geometric product. Altogether we have

$$\begin{aligned} Q \to \sum_i (\mathcal{C}(\boldsymbol{u}_i) \cdot Q)^2 &= \sum_i (\mathcal{C}(\boldsymbol{u}_i)\boldsymbol{B}Q)^2 = \\ &= \sum_i Q^T\boldsymbol{B}\mathcal{C}(\boldsymbol{u}_i)\mathcal{C}(\boldsymbol{u}_i)^T\boldsymbol{B}Q = Q^T\boldsymbol{P}Q, \quad \boldsymbol{u}_i \in \mathbb{U}. \end{aligned}$$

[7] This claim was proved in Example 1.1.4, i.e. $\boldsymbol{u}\boldsymbol{u} = \boldsymbol{u} \cdot \boldsymbol{u}$, when $\boldsymbol{u}$ is 1-vector.

where

$$\boldsymbol{P} = \sum_i \boldsymbol{B}\mathcal{C}(\boldsymbol{u}_i)\mathcal{C}(\boldsymbol{u}_i)^T\boldsymbol{B}, \quad \boldsymbol{u}_i \in \mathbb{U}. \tag{1.61}$$

To formulate the solution we have to decompose the matrix $\boldsymbol{P}$ into block matrix. If we try to compute $\boldsymbol{P}$ for general $\mathcal{C}(\boldsymbol{u})$, we get

$$\begin{aligned}
\boldsymbol{B}\mathcal{C}(\boldsymbol{u}) &= (-u_1u_2 \;\; -\tfrac{1}{2}(u_1^2-u_2^2) \;\; -\tfrac{1}{2}(u_1^2+u_2^2) \;\; u_1 \;\; u_2 \;\; -1 \;\; 0 \;\; 0)^T,\\
\mathcal{C}(\boldsymbol{u})^T\boldsymbol{B} &= (-u_1u_2 \;\; -\tfrac{1}{2}(u_1^2-u_2^2) \;\; -\tfrac{1}{2}(u_1^2+u_2^2) \;\; u_1 \;\; u_2 \;\; -1 \;\; 0 \;\; 0),\\
\boldsymbol{B}\mathcal{C}(\boldsymbol{u})\mathcal{C}(\boldsymbol{u})^T\boldsymbol{B} &= \begin{pmatrix} \boldsymbol{P}_0 & \boldsymbol{P}_1 & 0\\ \boldsymbol{P}_1^T & \boldsymbol{P}_c & 0\\ 0 & 0 & 0 \end{pmatrix},
\end{aligned} \tag{1.62}$$

where

$$\begin{aligned}
\boldsymbol{P}_0 &= \begin{pmatrix} u_1^2u_2^2 & \frac{1}{2}u_1u_2(u_1^2-u_2^2)\\ \frac{1}{2}u_1u_2(u_1^2-u_2^2) & \frac{1}{4}(u_1^2-u_2^2)^2 \end{pmatrix},\\
\boldsymbol{P}_1 &= \begin{pmatrix} \frac{1}{2}u_1u_2(u_1^2+u_2^2) & -u_1^2u_2 & -u_1u_2^2 & u_1u_2\\ \frac{1}{4}(u_1^4-u_2^4) & -\frac{1}{2}u_1(u_1^2-u_2^2) & -\frac{1}{2}u_2(u_1^2-u_2^2) & \frac{1}{2}(u_1^2-u_2^2) \end{pmatrix},\\
\boldsymbol{P}_c &= \begin{pmatrix} \frac{1}{4}(u_1^2+u_2^2)^2 & -\frac{1}{2}u_1(u_1^2+u_2^2) & -\frac{1}{2}u_2(u_1^2+u_2^2) & \frac{1}{2}(u_1^2+u_2^2)\\ -\frac{1}{2}u_1(u_1^2+u_2^2) & u_1^2 & u_1u_2 & -u_1\\ -\frac{1}{2}u_2(u_1^2+u_2^2) & u_1u_2 & u_2^2 & -u_2\\ \frac{1}{2}(u_1^2+u_2^2) & -u_1 & -u_2 & 1 \end{pmatrix}.
\end{aligned}$$

Let us write down block of the matrix B denoted as B_c,

$$\boldsymbol{B}_c = \begin{pmatrix} 0 & 0 & 0 & -1\\ 0 & 1 & 0 & 0\\ 0 & 0 & 1 & 0\\ -1 & 0 & 0 & 0 \end{pmatrix}. \tag{1.63}$$

The solution to the optimization problem (1.59), (1.60) if given by $Q = (\boldsymbol{w} \;\; \boldsymbol{v} \;\; 0)$, where $\boldsymbol{v}$ is 4-dimensional eigenvector corresponding to the minimal non-negative eigenvalue of

$$\boldsymbol{P}_{con} = \boldsymbol{B}_c(\boldsymbol{P}_c - \boldsymbol{P}_1\boldsymbol{P}_0^{-1}\boldsymbol{P}_1). \tag{1.64}$$

and $\boldsymbol{w}$ is a 2-dimensional vector of the form

$$\boldsymbol{w} = -\boldsymbol{P}_0^{-1}\boldsymbol{P}_1\boldsymbol{v}. \tag{1.65}$$

Example 1.2.10. For example let $\mathcal{C}(\boldsymbol{u}_i) \in \mathbb{U},\ i \in \{1,\dots,15\}$ be set of points defined by

$$\begin{aligned}
\mathbb{U} = \{&(-80.54474, -116.2646), (-38.52142, -116.2646), (-20.31128, -112.0623),\\
&(21.71204, -99.45526), (56.73151, -79.84436), (86.14789, -51.8288),\\
&(111.3619, -23.81323), (133.7744, 32.2179), (136.5759, 71.43967),\\
&(132.3735, 105.0583), (112.7626, 133.0739), (83.34631, 169.4941),\\
&(56.73151, 190.5058), (23.11279, 203.1129),\\
&(-14.70819, 214.319)\}.
\end{aligned}$$

For this set of points we get matrices $\boldsymbol{P}$ and $\boldsymbol{P}_{con}$

$$\boldsymbol{P} = \begin{pmatrix} 69718451 & -21911575 & 76249979 & -362046 & -535434 & 4785 & 0.000 & 0.000 \\ -21911575 & 107569190 & -93769285 & -101975 & 743144 & -4554 & 0.000 & 0.000 \\ 76249979 & -93769288 & 177287640 & -637409 & -1105190 & 11860 & 0.000 & 0.000 \\ -362046 & -101975 & -637409 & 7307 & 4785 & -53.37 & 0.000 & 0.000 \\ -535434 & 743144 & -1105190 & 4785 & 16414 & -34.65 & 0.000 & 0.000 \\ 4785 & -4554 & 11860 & -53.37 & -34.65 & 1.000 & 0.000 & 0.000 \\ 0.000 & 0.000 & 0.000 & 0.000 & 0.000 & 0.000 & 0.000 & 0.000 \\ 0.000 & 0.000 & 0.000 & 0.000 & 0.000 & 0.000 & 0.000 & 0.000 \end{pmatrix}$$

$$\boldsymbol{P}_{con} = \begin{pmatrix} -4512.815 & 35.059 & -19.514 & -0.579 \\ -391035.649 & 4964.133 & 3236.073 & -35.059 \\ -121031.621 & 3236.073 & 9019.504 & 19.514 \\ -45497331.090 & 391035.649 & 121031.621 & -4512.815 \end{pmatrix}$$

Vector $Q = (0.00033, -0.00087, 0.00556, -0.29094, 0.33530, -72.22643)$. In Figure 1.10 we see the result along with defined points. In Figure 1.11 we added one point to show how can one distant point change properties of the conic.

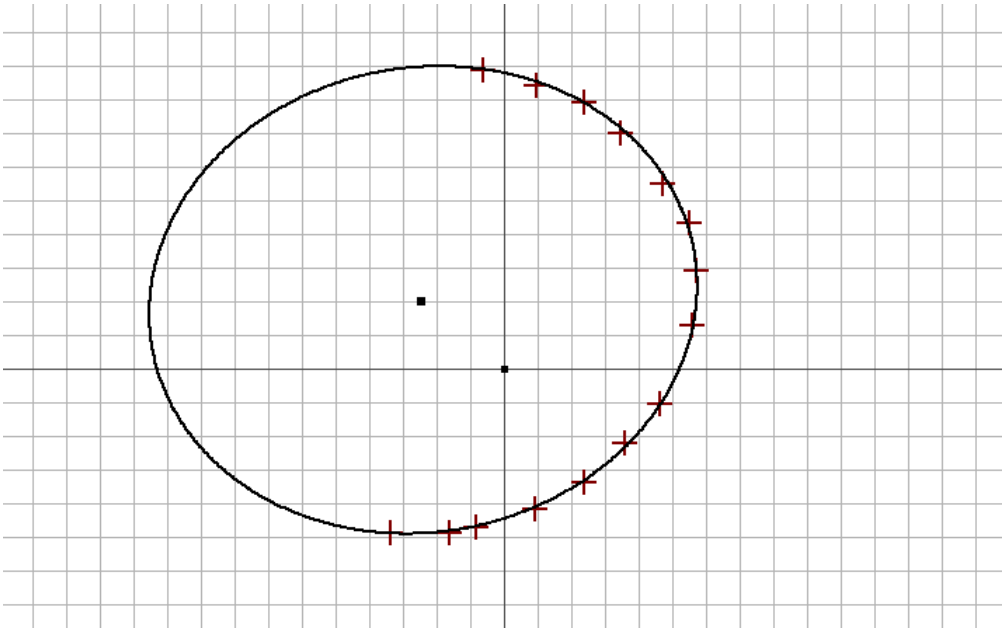


Figure 1.10: Ellipse fitted to set $\mathbb{U}$

In next chapter we will introduce computer application with implemented concepts shown in this chapter.

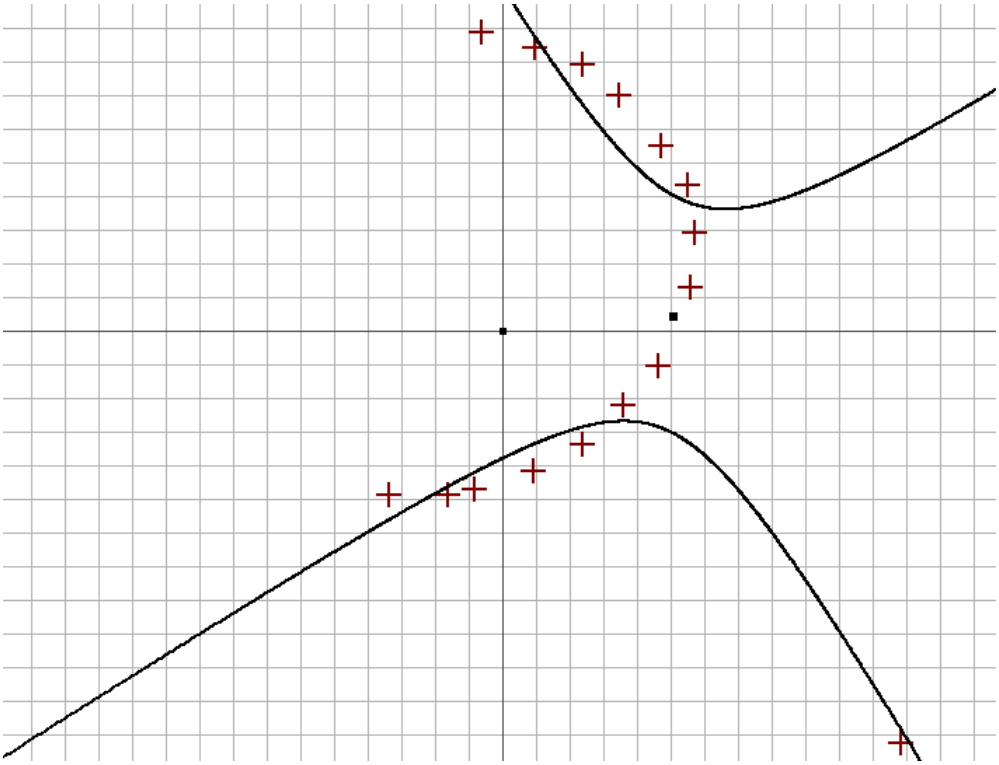


Figure 1.11: Hyperbola fitted to set $\mathbb{U}$ with one more distant point

2. Applications

2.1. Computing Engine

We did not discussed reason to do the transition between (1.30) and (1.31). Complications come with computing for example with a reverse to certain basis blades. Suppose a basis blade $\overline{\boldsymbol{n}}_\times \boldsymbol{n}_\times$, then reverse of this basis blade is $\boldsymbol{n}_\times \overline{\boldsymbol{n}}_\times$. Now, the question what basis blade is that reverse. We can answer that question by transiting these multivectors using (1.33),

$$\begin{aligned}
\overline{\boldsymbol{n}}_\times \boldsymbol{n}_\times &= \tfrac{1}{2}(-\boldsymbol{e}_1 + \boldsymbol{e}_8)(\boldsymbol{e}_1 + \boldsymbol{e}_8) = \tfrac{1}{2}(-1 - \boldsymbol{e}_1\boldsymbol{e}_8 + \boldsymbol{e}_8\boldsymbol{e}_1 - 1) = -1 - \boldsymbol{e}_1\boldsymbol{e}_8, \\
\boldsymbol{n}_\times \overline{\boldsymbol{n}}_\times &= \tfrac{1}{2}(\boldsymbol{e}_1 + \boldsymbol{e}_8)(-\boldsymbol{e}_1 + \boldsymbol{e}_8) = \tfrac{1}{2}(-1 + \boldsymbol{e}_1\boldsymbol{e}_8 - \boldsymbol{e}_8\boldsymbol{e}_1 - 1) = -1 + \boldsymbol{e}_1\boldsymbol{e}_8. \\
\boldsymbol{n}_\times \overline{\boldsymbol{n}}_\times &= -\overline{\boldsymbol{n}}_\times \boldsymbol{n}_\times - 2.
\end{aligned}$$

Without a property $\boldsymbol{e}_i\boldsymbol{e}_j = -\boldsymbol{e}_j\boldsymbol{e}_i$, $\forall i, j \in \{1, \ldots, p+q\} : i \neq j$ we would have to redefine a lot of relations in Section 1.1. In this section we introduce an approach to compute geometric product on general multivectors along with other important functions.

2.1.1. Defining Multivector in C#

Dots in the code can be understood in the same way as in mathematical text. There are two main approaches to structures in C# . We use classes, because it has no negative impact on functionality.

The code alone is quite long, thus we describe only the idea behind this approach. For the detailed and complete code see appendix with the program, where the source code is separately accessible. The C# code is not supposed to work on its own, it is called by Unity.

Because it is so much easier to work with positions of the basis vectors in their basis, we define the general multivector of GAC as 256-dimensional vector of coefficients $\boldsymbol{a} \in \mathbb{R}^{256}$. Each position $i \in \{1, \ldots, 256\}$ of this vector represents coefficient for basis blade $\boldsymbol{e}_{\mathbb{S}[i]}$, where $\mathbb{S} = \mathcal{P}_O(\{1, \ldots, 8\})$. Each multivector is initiated as $\boldsymbol{o} \in \mathbb{R}^{256}$;

Code 2.1: Multivector class

```
1 public class MVec
2 {
3   public double[] coef = new double[256]{0,0,...,0};
4 }
```

Then we can choose from several options, how we define any new multivector (`Mvec`).

Code 2.2: Creating new multivector

```
1 public MVec(double[] setcoef)
2 {
3   coef = setcoef;
4   //Create new MVec directly by 256-double
5 }
6 public MVec(int pos, double setcoef)
7 {
```

```
8     coef[pos] = setcoef;
9     //Create new MVec by setting coefficient to position
10  }
11  public MVec(int[] poss, double[] setcoefs)
12  {
13    int value = poss.Length;
14    if (value == coefs.Length)
15    {
16      for (int i = 0; i < value; i++)
17      {
18        coef[poss[i]] = coefs[i];
19      }
20    }
21    //Create new MVec by set of positions and corresponding
       coefficients
22  }
```

2.1.2. Performing Computations on Multivectors

All functions are related to the multivectors. We do not need to introduce the basis blades in the code, because all their properties can be derived from the position in their basis. Now we define connection of position to the power set we used in 1.1.6. The following functions are included in `public static class Compute`. The next function is implementation of $\{\mathcal{P}_O(\{1,\ldots,8\})[i]\}_i \ ,\forall i \in \{1,2,\ldots,256\}$.

Code 2.3: Obtaining the set of basis blade signatures from positions in their basis

```
1   static readonly int[,] setfromposition = new int[256,8]
2   {
3     { 0, 0, 0, 0, 0, 0, 0, 0},
4     { 1, 0, 0, 0, 0, 0, 0, 0},
```

```
258   { 1, 2, 3, 4, 5, 6, 7, 8},
259 }
```

This matrix is filled with all possible signatures ordered by the rule in Definition 1.1.6. This matrix is used in the next function.

Code 2.4: Basis blade signature from position

```
1   public static int[] SOP(int E)
2   {
3     int[] cache = new int[8];
4     for (int i = 0; i < 8; i++)
5     {
6       cache[i] = setfromposition[E - 1, i];
7     }
8     return cache;
9   }
10  //returns signature from position in the basis
```

Code 2.5: Position in basis from the basis blade signature

```
1  public static int POS(int[] Signature)
2  {
3    int translation = Signature[0]*(int)Mathf.Pow(10,7) + Signature
       [1]*(int)Mathf.Pow(10, 6) + Signature[2]*(int)Mathf.Pow(10, 5)
       + Signature[3]*(int)Mathf.Pow(10, 4) + Signature[4]*(int)Mathf.
       Pow(10, 3) + Signature[5]*(int)Mathf.Pow(10, 2) + Signature
       [6]*(int)Mathf.Pow(10, 1) + Signature[7];
4    int cache;
5    switch (translation)
6    {
7      case 0:
8        cache = 1;
9        break;
10     case 10000000:
11       cache = 2;
12       break;
```

```
772    case 12345678:
773      cache = 256;
774      break;
775    default:
776      cache = 1;
777      break;
778  }
779  return cache;
780 }
781 //computes temporary variable given by signature set and returns
      position
```

The reason why we need this connection to exist, is to be able to compute the grade only from the position in the basis. As in Definition 1.1.10, we compute a number of non-zero numbers in the basis blade signature set $\mathcal{P}_O(\{1,\dots,8\})[i],\ i \in \{1,\dots,256\}$[1].

Code 2.6: Grade from position of the basis blade in basis

```
1  public static int GetGrade(int position)
2  {
3    int[] cache = SOP(position);
4    int counter = 0;
5    for (int i = 0; i<8; i++)
6    {
7      if (cache[i] != 0)
8      {
9        counter++;
10     }
11   }
12   return counter;
13 }
```

[1]Grade of $\mathcal{P}_O(\{1,\dots,8\})[i]$ is just cardinality of that set, but in C# the cardinality is always 8 as you can see in Code 2.3.

```
14 //returns grade of position
```

As we mentioned that part of source code is written in MATLAB. The reason behind this is that we want to precompute all combinations of geometric product of two basis blades. A matrix $\boldsymbol{G} \in \{1, -1, 2, -2 \ldots, 256, -256\}^{256\times256}$, where

$$\boldsymbol{G}_{i,j} = \{l \in \mathbb{Z} : \ l = \mathrm{sgn}(\boldsymbol{E}_k)k\}, \ \ \text{where} \boldsymbol{E}_k = \boldsymbol{E}_i\boldsymbol{E}_j. \tag{2.1}$$

Thus, the geometric product can be computed using this matrix as follows, $\boldsymbol{E}_i\boldsymbol{E}_j = \mathrm{sgn}(\boldsymbol{G}_{i,j})\boldsymbol{E}_{|\boldsymbol{G}_{i,j}|}$. This approach reduces computation time, because in practice there are commonly hundreds or even thousands of geometric products computed in one step. We now explain algorithm for computation of geometric product. The code is written in MATLAB and is created by author.

Code 2.7: function code written in MATLAB, which returns basis blade signature together with its sign

```
1  function [result,signature] = geometricproduct(sgn1, sgn2, grade1,
       grade2)
2
3  changesignature = false;
4  n1 = grade1;
5  n2 = grade2;
6  k = 0;
7  r = 0;
8  for i = 1:8
9    for j = 1:8
10     if (sgn1(i) == sgn2(j)) && sgn1(i) ~= 0
11       k = k + 1;
12       if sgn1(i) >= 6
13         r = r + 1;
14       end
15     end
16   end
17 end
18 % computes number of common basis vectors k and number of -
     signature vectors
```

First part of the code computes number of common basis vectors in both basis blades and if this basis vector $\boldsymbol{e}_i$ has negative signature, i. e. $i \geq 6$.

```
19 if k > 0
20   for l = 1 : k
21     for i = 1:n1 + 1 - l
22       for j = l:n2
23         if  sgn1(i) > 0 && sgn2(j) > 0
24           if  sgn1(i) == sgn2(j)
25             if i ~= n1 + 1 - l
26               cache1 = sgn1(n1 + 1 - l);
27               sgn1(n1 + 1 - l) = sgn1(i);
28               sgn1(i) = cache1;
29               changesignature = ~ changesignature;
30             end
```

```
            if j ~= l
              cache2 = sgn2(l);
              sgn2(l) = sgn2(j);
              sgn2(j) = cache2;
              changesignature = ~ changesignature;
            end
          end
        end
      end
    end
  end
end
```

Second part of the code is ordering both basis blades, so they can cancel common basis vectors. Note that each change of position results in changing the sign.

```
if n1 > k && n2 > k
  for i = 1:n2 - k
    sgn1(n1 - k + i) = sgn2(k + i);
  end
elseif n1 == k
  for i = 1:n2 - k
    sgn1(i) = sgn2(k + i);
  end
end

if n1 + n2 - 2*k < 8
  for i = n1 + n2 - 2*k + 1:8
    sgn1(i) = 0;
  end
end

index = 0;

for i = 1:n1 + n2 - 2 * k
  min = 8;
  for j = i:n1 + n2 - 2 * k
    if sgn1(j) <= min
      min = sgn1(j);
      index = j;
    end
  end
  if index ~= i
    changesignature = ~changesignature;
    l = sgn1(i);
    sgn1(i) = sgn1(index);
    sgn1(index) = l;
  end
end
if changesignature == false
  signature = 1;
```

```
79 else
80   signature = -1;
81 end
82 signature = signature * (-1)^r;
83 result = sgn1;
84 end
```

The last part of the function performs the cancellation of common basis vectors and rewrites the signature of the first basis blade as the result. Note that the cancellation does not change the signature of the result, but rather this change is performed on line 82. Now, we write a script which generates C# source code.

Code 2.8: Using this code we implement matrix $\boldsymbol{G}$, which components are defined by (2.1), to C# source code

```
1  for i = 1:256
2    for j =1:256
3      [output2(i,j,:),outsgn(i,j)] = geometricproduct(output(i,:),
       output(j,:),outputdim(i),outputdim(j));
4      for k = 1:256
5        issame = true;
6        for l = 1:8
7          if output(k,l) ~= output2(i,j,l)
8            issame = false;
9            break;
10         end
11       end
12       if issame == true
13         output3(i,j) = outsgn(i,j) * k;
14       end
15     end
16   end
17 end
18 fprintf('static readonly int[,] Values = new int[256, 256]{');
19 for i = 1:256
20   fprintf('{');
21   for j =1:255
22     fprintf(' %.0f,',output3(i,j));
23   end
24   fprintf(' %.0f',output3(i,256));
25   fprintf('},\n');
26 end
27 fprintf('};');
```

Note that `output(i,:)` is signature of the basis blade $\boldsymbol{E}_i$, and `outputdim(i)` grade of $\boldsymbol{E}_i$.

Code 2.9: Part of the output given by Code 2.8

```
1  static readonly int[,] Values = new int[256, 256] {{ 1, 2, 3,
     ...},
2    { 2, 1, 10, 11, 12, 13, 14, 15, 16, 3, 4, 5, 6, 7, 8, 9, 38,
     ...},
```

```
256   {..., 15, -14, -13, 12, -11, 10, 9, -8, 7, 6, -5, 4, -3, 2, -1},
257 };
```

This concept of computing geometric product can be extended to general geometric algebra $\mathbb{G}_{p,q}$ and also can be implemented to C# . Now, lets implement grade projection of basis blade (1.1.10) to the some grade $k \in \{0, \dots, 8\}$.

Code 2.10: Function that returns true if basis blade with the certain position has targeted grade

```
1  static bool GradeP(int tgrade, int position)
2  {
3    int a = Compute.GetGrade(Mathf.Abs(position));
4    if(tgrade == a)
5    {
6      return true;
7    }
8    else
9    {
10     return false;
11   }
12 }
```

2.1.3. Geometric, Outer and Inner Product

Lets finish this section with functions that perform geometric, outer and inner products of 2 multivectors defined in Code 2.1. Recall that the geometric, outer and inner product of multivectors $\boldsymbol{U}, \boldsymbol{V} \in \mathbb{G}_{5,3}$ is computed by

$$\begin{aligned} \boldsymbol{UV} &= \sum_{i,j} u_i v_j (\boldsymbol{E}_i \boldsymbol{E}_j), && (1.14) \\ \boldsymbol{U} \wedge \boldsymbol{V} &= \sum_{i,j} u_i v_j (\boldsymbol{E}_i \wedge \boldsymbol{E}_j), && (1.21) \\ \boldsymbol{U} \cdot \boldsymbol{V} &= \sum_{i,j} u_i v_j (\boldsymbol{E}_i \cdot \boldsymbol{E}_j). && (1.20) \end{aligned}$$

Code 2.11: Function in C# computing geometric product of 2 multivectors

```
1  public static MVec GPMvec(MVec vec1, MVec vec2)
2  {
3    MVec result = new MVec();
4    List<int> vec1nonzero = new List<int>();
5    List<int> vec2nonzero = new List<int>();
6    int cache1;
7    double cache2;
8    for (int i = 0; i<256; i++)
9    {
10     if (vec1.coef[i] != 0)
11     {
12       vec1nonzero.Add(i);
13     }
```

```
14    }
15    for (int i = 0; i < 256; i++)
16    {
17      if (vec2.coef[i] != 0)
18      {
19        vec2nonzero.Add(i);
20      }
21    }
22    int[] vec1nonzer = vec1nonzero.ToArray();
23    int[] vec2nonzer = vec2nonzero.ToArray();
24    int n = vec1nonzer.Length;
25    int m = vec2nonzer.Length;
26    for (int i = 0; i < n; i++)
27    {
28      for (int j = 0; j < m; j++)
29      {
30        cache1 = Compute.Values[vec1nonzer[i], vec2nonzer[j]];
31        cache2 = Mathf.Sign(cache1) * vec1.coef[vec1nonzer[i]] *
     vec2.coef[vec2nonzer[j]];
32        result.coef[Mathf.Abs(cache1) - 1] += cache2;
33      }
34    }
35
36    return result;
37  }
```

In the case of outer and inner product, the lines 26-34 are changer as follows:

Code 2.12: Adjustments in Code 2.11 in case of computing outer product of 2 multivectors

```
32  int grade1;
33  int grade2;
34  for (int i = 0; i < n; i++)
35  {
36    for (int j = 0; j < m; j++)
37    {
38      grade1 = Compute.GetGrade(vec1nonzer[i] + 1);
39      grade2 = Compute.GetGrade(vec2nonzer[j] + 1);
40      cache1 = Compute.Values[vec1nonzer[i], vec2nonzer[j]];
41      if(GradeP(Mathf.Abs(grade1 + grade2), Mathf.Abs(cache1)))
42      {
43        cache2 = Mathf.Sign(cache1) * vec1.coef[vec1nonzer[i]] *
     vec2.coef[vec2nonzer[j]];
44        result.coef[Mathf.Abs(cache1) - 1] += cache2;
45      }
46    }
47  }
```

Code 2.13: Adjustments in Code 2.11 in case of computing inner product of 2 multivectors

```
32  int grade1;
33  int grade2;
```

```
34 for (int i = 0; i < n; i++)
35 {
36   for (int j = 0; j < m; j++)
37   {
38     grade1 = Compute.GetGrade(vec1nonzer[i] + 1);
39     grade2 = Compute.GetGrade(vec2nonzer[j] + 1);
40     cache1 = Compute.Values[vec1nonzer[i], vec2nonzer[j]];
41     if (GradeP(Mathf.Abs(grade1 - grade2), Mathf.Abs(cache1)))
42     {
43       cache2 = Mathf.Sign(cache1) * vec1.coef[vec1nonzer[i]] *
    vec2.coef[vec2nonzer[j]];
44       result.coef[Mathf.Abs(cache1) - 1] += cache2;
45     }
46   }
47 }
```

2.2. Displaying Conics

Before we show specific functions defining conics and transformations, we show transition for vectors. Recall that using (1.34) we can convert multivector in the terms of basis $\overline{\mathbb{R}}^{5,3}$ to the multivector in terms of basis $\overline{\mathbb{R}}^{5,3*}$. We specify this transition for vector $\boldsymbol{u} \in \mathbb{R}^{5,3} \subset \mathbb{G}_{5,3}$ as function $\phi : \mathbb{R}^{5,3} \subset \mathbb{G}_{5,3} \to \mathbb{R}^{5,3} \subset \mathbb{G}_{5,3}$

$$\begin{aligned} \boldsymbol{u} &= \left(u_1 \quad u_2 \quad u_3 \quad u_4 \quad u_5 \quad u_6 \quad u_7 \quad u_8\right), \\ \phi(\boldsymbol{u}) &= \\ &= \left(-u_1 + u_8 \quad -u_2 + u_7 \quad -u_3 + u_6 \quad u_4 \quad u_5 \quad \tfrac{1}{2}(u_3 + u_6) \quad \tfrac{1}{2}(u_2 + u_7) \quad \tfrac{1}{2}(u_1 + u_8)\right). \end{aligned} \tag{2.2}$$

Its inverse can be done using (1.33). Transition of vector $\boldsymbol{v} \in \mathbb{R}^{5,3} \subset \mathbb{G}_{5,3}$ in terms of basis $\overline{\mathbb{R}}^{5,3*}$ to the vector in terms of basis $\overline{\mathbb{R}}^{5,3}$ is defined by $\phi^{-1} : \mathbb{R}^{5,3} \subset \mathbb{G}_{5,3} \to \mathbb{R}^{5,3} \subset \mathbb{G}_{5,3}$,

$$\begin{aligned} \boldsymbol{v} &= \left(\overline{v}_\times \quad \overline{v}_- \quad \overline{v}_+ \quad v_1 \quad v_2 \quad v_+ \quad v_- \quad v_\times\right), \\ \phi^{-1}(\boldsymbol{v}) &= \\ &= \left(-\tfrac{1}{2}\overline{v}_\times + v_\times \quad -\tfrac{1}{2}\overline{v}_- + v_- \quad -\tfrac{1}{2}\overline{v}_+ + v_+ \quad v_1 \quad v_2 \quad \tfrac{1}{2}\overline{v}_+ + v_+ \quad \tfrac{1}{2}\overline{v}_- + v_- \quad \tfrac{1}{2}\overline{v}_\times + v_\times\right). \end{aligned} \tag{2.3}$$

Now we show a C# function that performs these functions ϕ^{-1} and ϕ.

Code 2.14: Implementation of function ϕ^{-1}

```
1 public static MVec C1VtoR53(MVec mvec)
2 {
3   MVec result = new MVec();
4   result.coef[1] = -mvec.coef[1] / 2f + mvec.coef[8];
5   result.coef[2] = -mvec.coef[2] / 2f + mvec.coef[7];
6   result.coef[3] = -mvec.coef[3] / 2f + mvec.coef[6];
7   result.coef[4] = mvec.coef[4];
8   result.coef[5] = mvec.coef[5];
```

```
9     result.coef[6] = mvec.coef[3] / 2f + mvec.coef[6];
10    result.coef[7] = mvec.coef[2] / 2f + mvec.coef[7];
11    result.coef[8] = mvec.coef[1] / 2f + mvec.coef[8];
12    return result;
13  }
```

Code 2.15: Implementation of function ϕ

```
1   public static MVec C1VtoR53star(MVec mvec)
2   {
3     MVec result = new MVec();
4     result.coef[1] = -mvec.coef[1] + mvec.coef[8];
5     result.coef[2] = -mvec.coef[2] + mvec.coef[7];
6     result.coef[3] = -mvec.coef[3] + mvec.coef[6];
7     result.coef[4] = mvec.coef[4];
8     result.coef[5] = mvec.coef[5];
9     result.coef[6] = mvec.coef[3] / 2f + mvec.coef[6] / 2f;
10    result.coef[7] = mvec.coef[2] / 2f + mvec.coef[7] / 2f;
11    result.coef[8] = mvec.coef[1] / 2f + mvec.coef[8] / 2f;
12    return result;
13  }
```

2.2.1. Conics in C#

In this subsection we implement IPNS and OPNS representations of conics to C# code. First, lets implement embedding of the point defined by (1.35),

Code 2.16: Implementation of embbeding point from 2D

```
1   public static MVec E2DtoG53(double u1, double u2)
2   {
3     MVec point = new MVec();
4     point.coef[3] = 1;
5     point.coef[4] = u1;
6     point.coef[5] = u2;
7     point.coef[6] = (u1 * u1 + u2 * u2) / 2;
8     point.coef[7] = (u1 * u1 - u2 * u2) / 2;
9     point.coef[8] = (u1 * u2);
10    return point;
11  }
```

From equations (1.45) and (1.48) we set IPNS of an ellipse and a hyperbola in terms of basis $\overline{\mathbb{R}}^{5,3^*}$:

Code 2.17: Implementation ellipse (type = 0) and hyperbola (type = 1)

```
1   public static MVec IPRConic(double a, double b, double c1, double
        C2, double angle, int type)
2   {
3     angle = angle * Mathf.Deg2Rad;
4     double gamma;
5     double alpha;
```

```
6     double beta;
7     float cos = Mathf.Cos(2 * (float)angle);
8     float sin = Mathf.Sin(2 * (float)angle);
9     MVec conic = new MVec();
10    if (type == 0)
11    {
12      gamma = (a * a + b * b);
13      alpha = (a * a - b * b);
14      beta = (2 * a * a * b * b);
15    }
16    else
17    {
18      gamma = (a * a - b * b);
19      alpha = (a * a + b * b) ;
20      beta = -(2 * a * a * b * b) ;
21    }
22    conic.coef[1] = -alpha * sin;
23    conic.coef[2] = -alpha * cos;
24    conic.coef[3] = 1 * gamma;
25    conic.coef[4] = (c1 * gamma - c1 * alpha * cos - C2 * alpha *
       sin);
26    conic.coef[5] = (C2 * gamma + C2 * alpha * cos - c1 * alpha *
       sin);
27    conic.coef[6] = ((c1 * c1 * gamma + C2 * C2 * gamma - beta - (c1
        * c1 - C2 * C2) * alpha * cos - 2 * c1 * C2 * alpha * sin) /
       2);
28    return conic;
29  }
```

In the case of OPNS representation we implement the conic spanned by 5 points $\boldsymbol{u}_1$, $\boldsymbol{u}_2$, $\boldsymbol{u}_3$, $\boldsymbol{u}_4$, $\boldsymbol{u}_5 \in \mathbb{R}^2$ as follows[2]

$$Q_O = \phi^{-1}(\mathcal{C}(\boldsymbol{u}_1)) \wedge \phi^{-1}(\mathcal{C}(\boldsymbol{u}_2)) \wedge \phi^{-1}(\mathcal{C}(\boldsymbol{u}_3)) \wedge \phi^{-1}(\mathcal{C}(\boldsymbol{u}_4)) \wedge \phi^{-1}(\mathcal{C}(\boldsymbol{u}_5)).$$

Code 2.18: Implementation of conic spanned by 5 points

```
1   public static MVec OPR5Points(Vector2 u1, Vector2 u2, Vector2 u3,
       Vector2 u4, Vector2 u5)
2   {
3     MVec result = OPMVec(C1VtoR53(E2DtoG53(u1.x,u1.y)), C1VtoR53(
       E2DtoG53(u2.x, u2.y)));
4     result = OPMVec(result, C1VtoR53(E2DtoG53(u3.x, u3.y)));
5     result = OPMVec(result, C1VtoR53(E2DtoG53(u4.x, u4.y)));
6     result = OPMVec(result, C1VtoR53(E2DtoG53(u5.x, u5.y)));
7     return result;
8   }
```

In the case of axes-aligned conic given by 4 points and one basis vector $\boldsymbol{n}_\times$,

$$Q_O^{al} = \phi^{-1}(\mathcal{C}(\boldsymbol{u}_1)) \wedge \phi^{-1}(\mathcal{C}(\boldsymbol{u}_2)) \wedge \phi^{-1}(\mathcal{C}(\boldsymbol{u}_3)) \wedge \phi^{-1}(\mathcal{C}(\boldsymbol{u}_4)) \wedge \phi^{-1}(\boldsymbol{n}_\times).$$

[2]The geometric(inner, outer) product accepts only 2 multivectors at the time, but due to associativity of the geometric product, the order does not matter.

Code 2.19: Implementation of axes-aligned conic spanned by 4 points

```
public static MVec OPRAL4Points(Vector2 u1, Vector2 u2, Vector2 u3
    , Vector2 u4)
{
  MVec result = OPMVec(C1VtoR53(E2DtoG53(u1.x, u1.y)), C1VtoR53(
   E2DtoG53(u2.x, u2.y)));
  result = OPMVec(result, C1VtoR53(E2DtoG53(u3.x, u3.y)));
  result = OPMVec(result, C1VtoR53(E2DtoG53(u4.x, u4.y)));
  result = OPMVec(result, C1VtoR53(new MVec(8,1)));
  return result;
}
```

In the case of circle spanned by 3 points and basis vector of GAC representing infinity $\boldsymbol{n}_\times, \boldsymbol{n}_-$,

$$C_O = \phi^{-1}(\mathcal{C}(\boldsymbol{u}_1)) \wedge \phi^{-1}(\mathcal{C}(\boldsymbol{u}_2)) \wedge \phi^{-1}(\mathcal{C}(\boldsymbol{u}_3)) \wedge \phi^{-1}(\boldsymbol{n}_-) \wedge \phi^{-1}(\boldsymbol{n}_\times). \tag{2.4}$$

Code 2.20: Implementation of circle spanned by 3 points

```
public static MVec OPRC3Points(Vector2 u1, Vector2 u2, Vector2 u3)
{
  MVec result = OPMVec(C1VtoR53(E2DtoG53(u1.x, u1.y)), C1VtoR53(
   E2DtoG53(u2.x, u2.y)));
  result = OPMVec(result, C1VtoR53(E2DtoG53(u3.x, u3.y)));
  result = OPMVec(result, C1VtoR53(new MVec(7, 1)));
  result = OPMVec(result, C1VtoR53(new MVec(8, 1)));
  return result;
}
```

As we will perform conic transformations on IPNS representation, we implement dual operator as follows. From (1.49) and Example 1.1.6,

$$Q_I = (Q_O \wedge \phi^{-1}(\overline{\boldsymbol{n}}_-) \wedge \phi^{-1}(\overline{\boldsymbol{n}}_\times))(-\boldsymbol{I}). \tag{2.5}$$

Code 2.21: Implementation of dual operator for OPNS representation

```
public static MVec OPRDual(MVec OPR)
{
  MVec PseudoS = new MVec();
  PseudoS.coef[255] = -1;
  MVec resutl = OPMVec(OPR, C1VtoR53(new MVec(2, 1)));
  resutl = OPMVec(resutl, C1VtoR53(new MVec(1, 1)));
  resutl = GPMvec(resutl, PseudoS);
  return resutl;
}
```

2.2.2. Transformations in C#

This subsection finishes implementations of $\mathbb{G}_{5,3}$ functionalities to C# source code. Clockwise rotation by angle α done in $\mathbb{G}_{5,3}$ is given by rotor $\boldsymbol{R}$ and its reverse $\widetilde{\boldsymbol{R}}$.

$$\begin{aligned}\boldsymbol{R} &= (\cos(\tfrac{\alpha}{2}) + \sin(\tfrac{\alpha}{2})\boldsymbol{e}_4 \wedge \boldsymbol{e}_5)\Big(\big(\cos(\alpha) + \sin(\alpha)(\phi^{-1}(\overline{\boldsymbol{n}}_\times) \wedge \\ &\quad \wedge \phi^{-1}(\boldsymbol{n}_-))\big) \wedge \big(\cos(\alpha) - \sin(\alpha)(\phi^{-1}(\overline{\boldsymbol{n}}_-) \wedge \phi^{-1}(\boldsymbol{n}_\times))\big)\Big).\end{aligned} \tag{2.6}$$

Code 2.22: Implementation of rotor

```
public static MVec Rotor(double alpha)
{
  double angle = alpha* Mathf.Deg2Rad;
  MVec Rplus = new MVec();
  Rplus.coef[0] += Mathf.Cos((float)angle / 2f);
  Rplus.coef[27] += Mathf.Sin((float)angle / 2f);
  MVec R1 = MByConst(Mathf.Sin((float)angle), OPMVec(C1VtoR53(new
   MVec(1, 1)), C1VtoR53(new MVec(7, 1))));
  R1.coef[0] += Mathf.Cos((float)angle);
  MVec R2 = MByConst(-Mathf.Sin((float)angle), OPMVec(C1VtoR53(new
    MVec(2, 1)), C1VtoR53(new MVec(8, 1))));
  R2.coef[0] += Mathf.Cos((float)angle);
  MVec result = OPMVec(R1, R2);
  result = GPMvec(Rplus, result);
  ahoj(result);
  return result;
}
```

Translation by a_1 in direction $\boldsymbol{e}_1$ and a_2 in direction $\boldsymbol{e}_2$ is given by translator $\boldsymbol{T} = \boldsymbol{T}_{\boldsymbol{e}_1}\boldsymbol{T}_{\boldsymbol{e}_2}$ and its reverse $\widetilde{\boldsymbol{T}}$, where

$$\begin{aligned}\boldsymbol{T}_{\boldsymbol{e}_1} &= \big((1 - \tfrac{1}{2}a_1\boldsymbol{e}_4 \wedge (\boldsymbol{e}_3 + \boldsymbol{e}_6))(1 - \tfrac{1}{2}a_1\boldsymbol{e}_4 \wedge (\boldsymbol{e}_2 + \boldsymbol{e}_7) + \tfrac{1}{4}a_1^2(\boldsymbol{e}_3 + \boldsymbol{e}_6) \wedge (\boldsymbol{e}_2 + \boldsymbol{e}_7))\big) \\ &\quad (1 - \tfrac{1}{2}a_1\boldsymbol{e}_5 \wedge (\boldsymbol{e}_3 + \boldsymbol{e}_6)). \\ \boldsymbol{T}_{\boldsymbol{e}_2} &= \big((1 - \tfrac{1}{2}a_2\boldsymbol{e}_5 \wedge (\boldsymbol{e}_3 + \boldsymbol{e}_6))(1 - \tfrac{1}{2}a_2\boldsymbol{e}_5 \wedge (\boldsymbol{e}_2 + \boldsymbol{e}_7) + \tfrac{1}{4}a_2^2(\boldsymbol{e}_3 + \boldsymbol{e}_6) \wedge (\boldsymbol{e}_2 + \boldsymbol{e}_7))\big) \\ &\quad (1 - \tfrac{1}{2}a_2\boldsymbol{e}_4 \wedge (\boldsymbol{e}_3 + \boldsymbol{e}_6)).\end{aligned} \tag{2.7}$$

The following code construct a translator in the direction $\boldsymbol{e}_1$, translator in the direction $\boldsymbol{e}_2$ is implemented analogously.

Code 2.23: Implementation of translator in direction $\boldsymbol{e}_1$

```
public static MVec Translatore1(double a1)
{
  MVec cache;
  MVec Tplus = MByConst(-a1 / 2f, OPMVec(C1VtoR53(new MVec(4, 1)),
    C1VtoR53(new MVec(6, 1))));
  Tplus.coef[0] += 1;
  MVec Tminus = MByConst(-a1 / 2f, OPMVec(C1VtoR53(new MVec(4, 1))
   , C1VtoR53(new MVec(7, 1))));
```

```
7     cache = MByConst(a1 * a1 / 4f, OPMVec(C1VtoR53(new MVec(6, 1)),
       C1VtoR53(new MVec(7, 1))));
8     Tminus = Plus(Tminus, cache);
9     Tminus.coef[0] += 1;
10    MVec Tx = MByConst(-a1 / 2f, OPMVec(C1VtoR53(new MVec(5, 1)),
       C1VtoR53(new MVec(8, 1))));
11    Tx.coef[0] += 1;
12    MVec result = GPMvec(Tplus, Tminus);
13    result = GPMvec(result, Tx);
14    return result;
15 }
```

Scale by α is given by scalor $\boldsymbol{S}$ and its reverse $\widetilde{\boldsymbol{S}}$.

$$\begin{aligned}\boldsymbol{S} \;=\; & \left(\left(\tfrac{\alpha+1}{2\sqrt{\alpha}} + \tfrac{\alpha-1}{2\sqrt{\alpha}}\left(\tfrac{1}{2}(-\boldsymbol{e}_3+\boldsymbol{e}_6)\wedge(\boldsymbol{e}_3+\boldsymbol{e}_6)\right)\right)\left(\tfrac{\alpha+1}{2\sqrt{\alpha}} + \tfrac{\alpha-1}{2\sqrt{\alpha}}\left(\tfrac{1}{2}(-\boldsymbol{e}_2+\boldsymbol{e}_7)\wedge(\boldsymbol{e}_2+\boldsymbol{e}_7)\right)\right)\right) \\ & \left(\tfrac{\alpha+1}{2\sqrt{\alpha}} + \tfrac{\alpha-1}{2\sqrt{\alpha}}\left(\tfrac{1}{2}(-\boldsymbol{e}_1+\boldsymbol{e}_8)\wedge(\boldsymbol{e}_1+\boldsymbol{e}_8)\right)\right).\end{aligned} \tag{2.8}$$

Code 2.24: Implementation of scalor

```
1  public static MVec Scalor(double a)
2  {
3     MVec Splus = MByConst(((float)a - 1)/(2*Mathf.Sqrt((float)a)),
       OPMVec(C1VtoR53(new MVec(3, 1)), C1VtoR53(new MVec(6, 1))));
4     Splus.coef[0] += (a + 1) / (2 * Mathf.Sqrt((float)a));
5     MVec Smin = MByConst((a - 1) / (2 * Mathf.Sqrt((float)a)),
       OPMVec(C1VtoR53(new MVec(2, 1)), C1VtoR53(new MVec(7, 1))));
6     Smin.coef[0] += (a + 1) / (2 * Mathf.Sqrt((float)a));
7     MVec Sx = MByConst((a - 1) / (2 * Mathf.Sqrt((float)a)), OPMVec(
       C1VtoR53(new MVec(1, 1)), C1VtoR53(new MVec(8, 1))));
8     Sx.coef[0] += (a + 1) / (2 * Mathf.Sqrt((float)a));
9     MVec result = GPMvec(Splus, Smin);
10    result = GPMvec(result, Sx);
11    ahoj(result);
12    return result;
```

2.2.3. Conic Properties

We will finish this section with getting conic properties from its general representation. As it is not the goal of this thesis, we will not explain theory behind the equations. From conic properties and parametric equations for an ellipse and hyperbola we can obtain set of points (pixels), which will be displayed as a graphical output. To get more knowledge

about conics see [2, 11, 14]. Recall the general conic section equation and its matrix representation Q with its determinant $|Q|$ in the form

$$Au_1^2 + 2Bu_1u_2 + Cu_2^2 + 2Du_1 + 2Eu_2 + F = 0, \qquad Q = \begin{pmatrix} A & B & D \\ B & C & E \\ D & E & F \end{pmatrix},$$

$$|Q| = \begin{vmatrix} A & B & D \\ B & C & E \\ D & E & F \end{vmatrix} = ACF + 2BDE - CD^2 - AE^2 - B^2F.$$

From the matrix representation Q of conic sections we can derive their properties. If $|Q| = 0$ we say that the conic is degenerate (intersected lines, parallel lines). If $|Q| \neq 0$ so that Q is not degenerate and we are able to see what type of conic section it is. Computing determinant of a sub-matrix $Q_{2\times2}$ of Q

$$Q_{2\times2} = \begin{pmatrix} A & B \\ B & C \end{pmatrix}, \quad |Q_{2\times2}| = \begin{vmatrix} A & B \\ B & C \end{vmatrix} = AC - B^2,$$

gives information which type of conic section it is. Conic Q is an ellipse $\iff |Q_{2\times2}| > 0$, parabola if $|Q_{2\times2}| = 0$ and hyperbola if $|Q_{2\times2}| < 0$. We care about ellipses and hyperbolas, because other types of conics are unlikely to occur in conic fitting algorithm. Ellipses and hyperbolas are central conics. Their center $(c_1, c_2) \in \mathbb{R}^2$ is computed as

$$\begin{pmatrix} c_1 \\ c_2 \end{pmatrix} = \begin{pmatrix} A & B \\ B & C \end{pmatrix}^{-1} \begin{pmatrix} -D \\ -E \end{pmatrix} = \begin{pmatrix} \frac{BE-CD}{AC-B^2} \\ \frac{DB-AE}{AC-B^2} \end{pmatrix}.$$

Counterclockwise angle $\theta \in (-\frac{1}{4}\pi, \frac{3}{4}\pi\rangle$ of rotation round $\boldsymbol{e}_1$ axis for ellipses is given by

$$\theta = \begin{cases} \frac{1}{4}\pi, & A = C \text{ and } \operatorname{sign}(B) \neq \operatorname{sign}(|Q_{2\times2}|), \\ \frac{3}{4}\pi, & A = C \text{ and } \operatorname{sign}(B) = \operatorname{sign}(|Q_{2\times2}|), \\ \frac{1}{2}\arctan(\frac{2B}{A-C}), & A < C, \\ \frac{1}{2}\pi + \frac{1}{2}\arctan(\frac{2B}{A-C}), & A > C. \end{cases}$$

And counterclockwise angle $\theta \in (-\frac{1}{4}\pi, \frac{3}{4}\pi\rangle$ of rotation around $\boldsymbol{e}_1$ axis for hyperbolas is given by

$$\theta = \begin{cases} \frac{1}{4}\pi, & A = C \text{ and } \operatorname{sign}(B) \neq \operatorname{sign}(|Q_{2\times2}|), \\ \frac{3}{4}\pi, & A = C \text{ and } \operatorname{sign}(B) = \operatorname{sign}(|Q_{2\times2}|), \\ \frac{1}{2}\arctan(\frac{2B}{A-C}), & A > C, \\ \frac{1}{2}\pi + \frac{1}{2}\arctan(\frac{2B}{A-C}), & A < C. \end{cases}$$

Note that non-degenerate, non-parabolic conic with angle $\theta \in (-\frac{1}{4}\pi, \frac{3}{4}\pi\rangle$ is the same conic as conic with angle $\theta + \pi$. To find the semi-axes lengths we have to compute eigenvalues $\lambda_{1,2}$ of the matrix $Q_{2\times2}$ first.

$$|Q_{2\times2}| = \begin{vmatrix} A-\lambda & B \\ B & C-\lambda \end{vmatrix} = 0, \qquad \lambda^2 - (A+C)\lambda + AC - B^2 = 0$$

$$\lambda_{1,2} = \frac{(A+C) \pm \sqrt{(A+C)^2 - 4(AC - B^2)}}{2}$$

Then the centralized conic with $u_1' = u_1 - c_1$ and $u_2' = u_2 - c_2$, where u_1, u_2 are points lying on conic, can be rewritten to its standard form and thus we can get semi-axes length a, b.

$$\lambda_1 {u_1'}^2 + \lambda_2 {u_2'}^2 = -\frac{|Q|}{|Q_{2\times 2}|} \quad \Longrightarrow \quad -\frac{|Q_{2\times 2}|\lambda_1 {u_1'}^2}{|Q|} - \frac{|Q_{2\times 2}|\lambda_2 {u_2'}^2}{|Q|} = 1.$$

If Q is hyperbola, then $\lambda_{1,2}$ have opposite sign. Now the centralized axes-aligned ellipse, hyperbola has following equation:

$$\frac{{u_1'}^2}{a^2} + \frac{{u_2'}^2}{b^2} = 1, \quad \frac{{u_1'}^2}{a^2} - \frac{{u_2'}^2}{b^2} = 1.$$

The squared semi-axes length a, b for ellipses are as follows. Note, that these equations hold for $|Q| > 0$. If $|Q| < 0$ then we simply multiply Q by -1.

$$\begin{aligned} a^2 &= -\frac{|Q|}{|Q_{2\times 2}|\lambda_1} = -\frac{2(ACF + 2BDE - CD^2 - AE^2 - B^2F)}{(AC - B^2)((A + C) + \sqrt{(A + C)^2 - 4(AC - B^2)})}, \\ b^2 &= -\frac{|Q|}{|Q_{2\times 2}|\lambda_2} = -\frac{2(ACF + 2BDE - CD^2 - AE^2 - B^2F)}{(AC - B^2)((A + C) - \sqrt{(A + C)^2 - 4(AC - B^2)})}. \end{aligned}$$

The squared semi-axes length a, b for hyperbolas are as follows:

$$\begin{aligned} a^2 &= -\frac{|Q|}{|Q_{2\times 2}|\lambda_1} = -\frac{2(ACF + 2BDE - CD^2 - AE^2 - B^2F)}{(AC - B^2)((A + C) + \sqrt{(A + C)^2 - 4(AC - B^2)})}, \\ b^2 &= \frac{|Q|}{|Q_{2\times 2}|\lambda_2} = \frac{2(ACF + 2BDE - CD^2 - AE^2 - B^2F)}{(AC - B^2)((A + C) - \sqrt{(A + C)^2 - 4(AC - B^2)})}. \end{aligned}$$

The parametric equation for the ellipse with counterclockwise rotation θ, centered in (c_1, c_2) and with semi-axes lengths a, b is given by

$$\begin{pmatrix} u_1 \\ u_2 \end{pmatrix} = \begin{pmatrix} c_1 + a\cos(\alpha)\cos(\theta) - b\sin(\alpha)\sin(\theta) \\ c_2 + a\cos(\alpha)\sin(\theta) + b\sin(\alpha)\cos(\theta) \end{pmatrix} \quad \alpha \in \langle 0, 2\pi). \tag{2.9}$$

Analogously for a hyperbola with counterclockwise rotation θ, centered in (c_1, c_2) and with semi-axes lengths a, b is given by

$$\begin{pmatrix} u_1 \\ u_2 \end{pmatrix} = \begin{pmatrix} c_1 + a\cosh(\alpha)\cos(\theta) - b\sinh(\alpha)\sin(\theta) \\ c_2 + a\cosh(\alpha)\sin(\theta) + b\sinh(\alpha)\cos(\theta) \end{pmatrix} \quad \alpha \in \mathbb{R}. \tag{2.10}$$

2.3. Ellipse Extraction

Suppose a camera, lying somewhere on the tube axis, with a point light source illuminating some neighborhood. Let the camera be rotated, so that it is oriented to same direction as a tangent of the axis. Then the camera can capture 2 types of images (Figures 2.1, 2.2). If we use certain image filters, we can extract a set of points (pixels of the image) highlighting a character of the tube in certain distance[3] in front of the camera. These points (pixels) are using conic fitting algorithm (introduced in Subsection 1.2.5) to get conic (in this case our aim is to always get an ellipse) in IPNS representation (vector in $\mathbb{R}^{5,3} \subset \mathbb{G}_{5,3}$). Matrix form of that ellipse is then obtained using (1.41) and its properties from equations defined in Subsection 2.2.3. From the ellipse properties we can further estimate a navigation trajectory through the tube. Note that in this Section we will use three kind of points; point of image (pixel with coordinates i, j) is denoted by $\boldsymbol{I}_{i,j}$[4], point projected onto rectangle (2D subspace of Euclidean space $\mathbb{R}^3$) is denoted by $\overline{\boldsymbol{u}}_{proj}$ (and its scaled variation $\boldsymbol{u}_{proj}$), and point from Euclidean space $\mathbb{R}^3$ is denoted by $\boldsymbol{u}$.

In the first Subsection we briefly introduce principle of creating 3D object in graphical software. Then we analyze the camera view in the dark tube (illuminated by a point source of light in the camera's position) and define a highlight matrices carrying points for the conic fitting algorithm. By means of projective geometry in computer graphics, we analyze and try to restore the tube axis using gained ellipses from conic fitting algorithm from different points in the tube. In this Section we commonly use term point, by what we mean vector that goes from origin to that point.

In the tube a plane intersection with the tube results in a circle with and from our camera position can be viewed as a circle or an ellipse. In Figures 2.5, 2.6 we can see discussed situation, where green plane is set by point of the axis and by its tangent in that point. We can already see the connection between the curved tube and ellipse observed from camera position.

2.3.1. Tube generation

In this Subsection we briefly introduce computer graphic software (Unity) on tube generation. Tube assumed in this thesis can be an object that consist of 2 types of segments; cylinder and torus. Tube can have only constant diameter and its surface has to be continuous and transition between the cylinder and the torus has to be smooth. Then the tube axis is either a line or part of a circle, which follow each other smoothly. The Figure 2.3 shows the principle of creating the tube mesh. Note that vectors $\boldsymbol{p}, \boldsymbol{t}_c$ are normalized. Object in 3D computer graphics consist of triangles, see [13]. In Unity, first we have to create a set of points, which lie on the tube. After that we create a set which connects the points in particular way. As we can see on the right part of Figure 2.3, the triangle is created by certain 3 points. Important is the order of points, because it carries also information about the normal of the triangle. For example taking triangle indexed by $\{1, 0, m\}$, results in downward facing normal and triangle indexed by $\{0, 1, m\}$ result in upward facing normal.

[3] We suppose that the filter will extract pixels of projected tube points that are in a similar distance to the camera.

[4] Symbol $\boldsymbol{I}$ denotes pseudoscalar in terms of geometric algebra. However, in this Section, we do not use geometric algebra terminology, so by $\boldsymbol{I}$ we mean an image.

Figure 2.1: Camera view in a straight tube

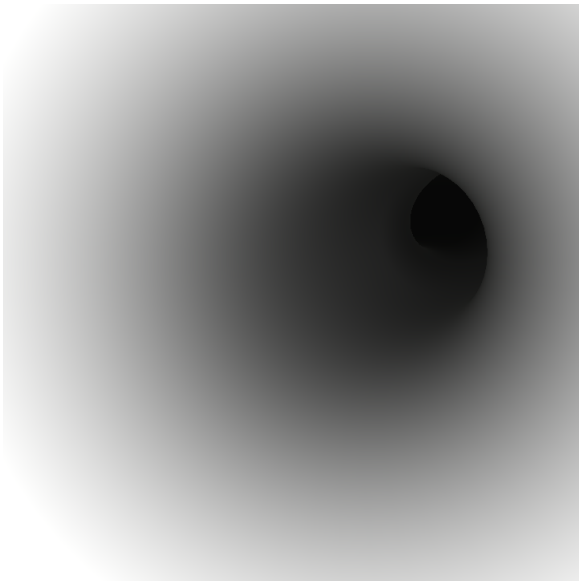


Figure 2.2: Camera view in a curved tube

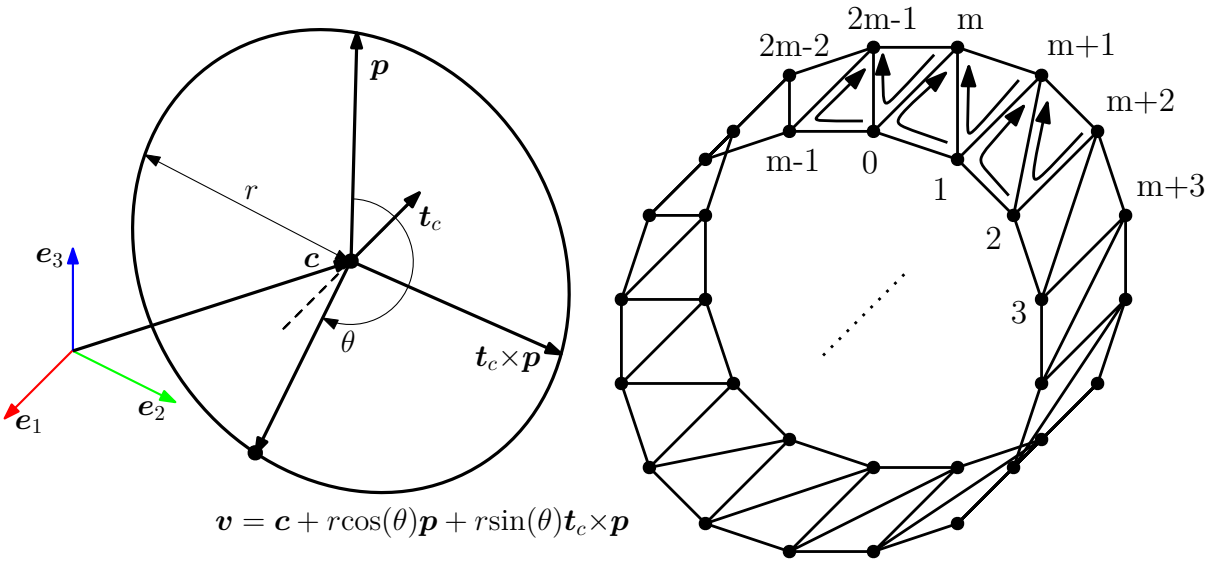


Figure 2.3: A way to create a tube mesh in the computer geometry

Definition 2.3.1. Let $\{\boldsymbol{c}_i\}_i, \boldsymbol{c}_i \in \mathbb{R}^3, i = 1, \ldots, n$ be a set of points of the tube axis $\boldsymbol{c}$ and let vector $\boldsymbol{t}_{ci} \in \mathbb{R}^3$ be tangent to the axis at the point $\boldsymbol{c}_i$. Then the set of points that lie on a tube can be obtained from the following $n \times m$ point set

$$\boldsymbol{V}_{i(m-1)+j+1} = \boldsymbol{c}_i + r\cos(\frac{j}{m})\boldsymbol{p}_i + r\sin(\frac{j}{m})\boldsymbol{t}_{ci} \times \boldsymbol{p}_i,$$

where $r \in \mathbb{R}^+$, $\boldsymbol{p}_i \in \mathbb{R}^3$ [5] is orthogonal to $\boldsymbol{t}_{ci}$, and $j = 0, \ldots, m-1$. This set is called set of vertices.

Definition 2.3.2. Let $\boldsymbol{V}_{n\times m}$ be a set of vertices of the tube, where n is a number of points that lie on the axis and m is a number of points lying on single circle of the tube.

[5] We can choose the initial point $\boldsymbol{p}_1$ arbitrarily, but we have to be consistent with the rotation of $\boldsymbol{p}_i$. For each $\boldsymbol{t}_{ci}$ we find rotation angles to $\boldsymbol{t}_{ci+1}$. These rotations are applied to $\boldsymbol{p}_i$. This results in a smooth mesh.

Then the set of triangles, that connect the vertices of the tube is defined as a set of indexes $\boldsymbol{T}_k, k = 1, \ldots, 6(n-1)m$,

$$\begin{aligned}
\boldsymbol{T}_{6((i-1)(m-1)+j)+1} &= (i-1)(m-1)+j+1 \\
\boldsymbol{T}_{6((i-1)(m-1)+j)+2} &= (i-1)(m-1)+j \\
\boldsymbol{T}_{6((i-1)(m-1)+j)+3} &= i(m-1)+j \\
\boldsymbol{T}_{6((i-1)(m-1)+j)+4} &= i(m-1)+j+1 \\
\boldsymbol{T}_{6((i-1)(m-1)+j)+5} &= (i-1)(m-1)+j+1 \\
\boldsymbol{T}_{6((i-1)(m-1)+j)+6} &= i(m-1)+j
\end{aligned} \qquad i \in \{1, \ldots, n-1\}, j \in \{1, \ldots, m-2\}. \tag{2.11}$$

$$\begin{aligned}
\boldsymbol{T}_{6((i-1)(m-1)+j)+1} &= (i-1)m \\
\boldsymbol{T}_{6((i-1)(m-1)+j)+2} &= (i-1)m+j \\
\boldsymbol{T}_{6((i-1)(m-1)+j)+3} &= im+j \\
\boldsymbol{T}_{6((i-1)(m-1)+j)+4} &= im \\
\boldsymbol{T}_{6((i-1)(m-1)+j)+5} &= (i-1)m \\
\boldsymbol{T}_{6((i-1)(m-1)+j)+6} &= im+j
\end{aligned} \qquad i \in \{1, \ldots, n-1\}, j = m-1. \tag{2.12}$$

There are $2(n-1)m-1$ triangles, where k-th triangle can be obtained from the set of vertices and set of triangles as follows, either from set of indexes (the k-th triangle is indexed by) $\{\boldsymbol{T}_{3k+1}, \boldsymbol{T}_{3k+2}, \boldsymbol{T}_{3k+3}\}$, or from set of vertices (the k-th triangle is generated by vertices) $\{\boldsymbol{V}_{\boldsymbol{T}_{3k+1}}, \boldsymbol{V}_{\boldsymbol{T}_{3k+2}}, \boldsymbol{V}_{\boldsymbol{T}_{3k+3}}\}$.

Six points are defining one rectangle. Equation (2.12) completes the last missing rectangle to (2.11). For example if we look at Figure 2.3, we can clearly get first 2 triangles from $\{\{\boldsymbol{T}_i\}, i = 1, \ldots, 6\} = \{1, 0, m, m+1, 1, m\}$ and the last two triangles from $\{\{\boldsymbol{T}_j\}, j = 6(m-1)+1, \ldots, 6(m-1)+6\} = \{0, m-1, 2m-1, m, 0, 2m-1\}$.

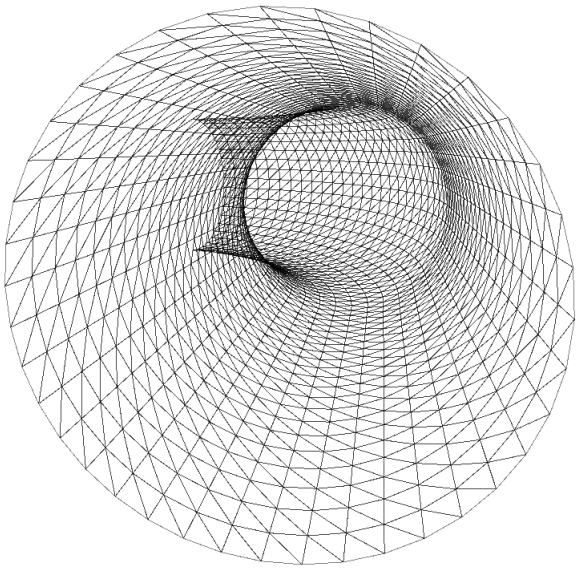


Figure 2.4: Wire frame of tube in Unity[6]

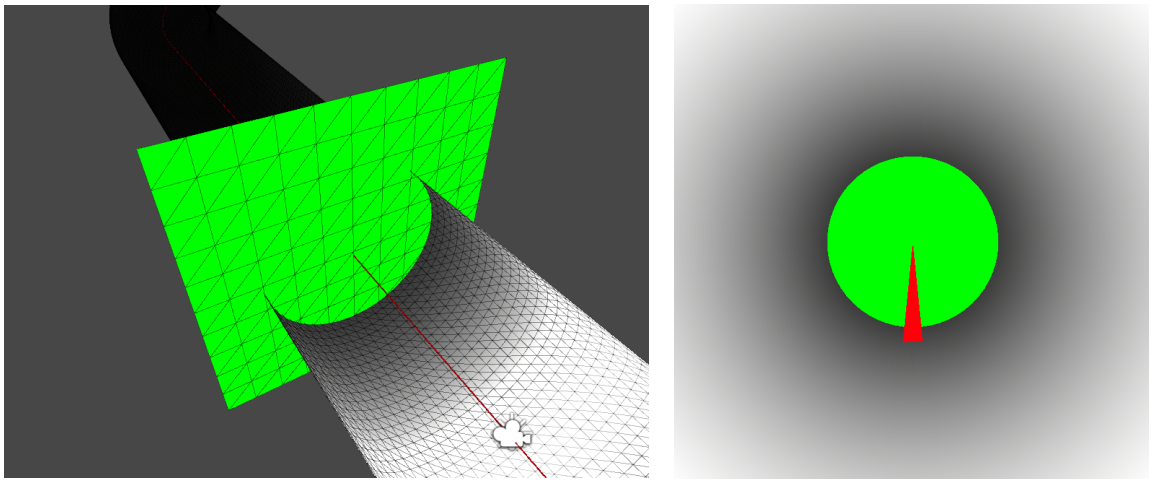


Figure 2.5: Plane intersection example in a straight tube.

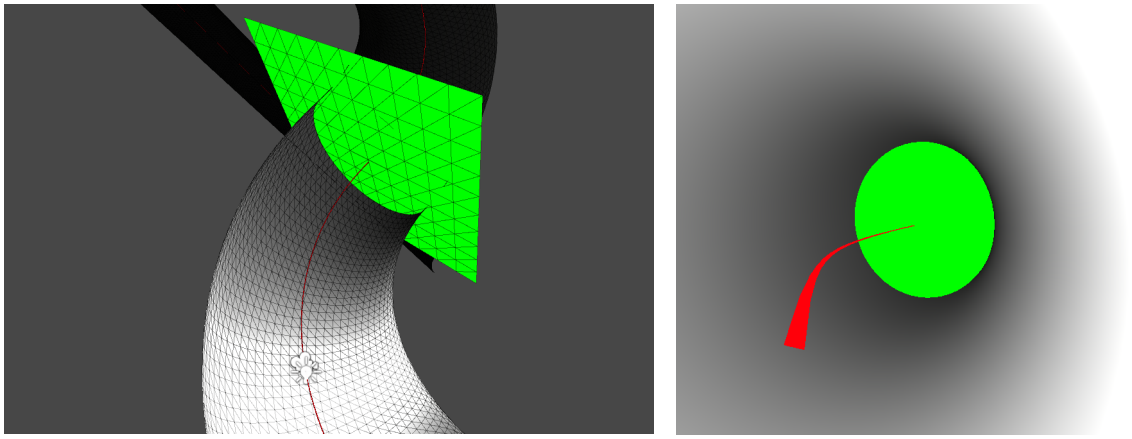


Figure 2.6: Plane intersection example in a curved tube

2.3.2. Pixels Selection

In this subsection we introduce two methods how to get a set of points, which we will fill the conic fitting algorithm. As we discussed in the introduction to this Section, suppose a camera that lies on the tube axis and in addition, it is pointed in the direction of a tangent to the axis. If the conic, produced by the fitting algorithm, is centered in the middle of the image, then the tube is straight in some neighborhood in front of the camera. If the center is not in the middle of the image, then the tube is curved in some distance. An example for the first case is Figure 2.1 and for the second see Figure 2.2. Question is how to find a set of points that can generate ellipses shown in Figures 2.5, 2.6.

The first algorithm for finding such set is taking points with certain brightness (color value). Let $\boldsymbol{I}$ be considered as a cameras screen[7]. Unity has function `2DTexture.GetPixel(i,j).grayscale`, which can be understood as the brightness (converts a color value to a gray scale) of the pixel, because it converts ARGB[8] format into

[6]Note that Unity uses inverted $\boldsymbol{e}_1, \boldsymbol{e}_2$ axis, so compared to the left part of Figure 2.3, the points are ordered counterclockwise.

[7]As a screen we consider a matrix (image) $\boldsymbol{I}$ with the following properties; $\boldsymbol{I}_{width}(\boldsymbol{I}_{height})$ is a number of pixels in image's horizontal(vertical) layer, $\boldsymbol{I}_{i,j}$ is a color value of the pixel i, j. However, we consider $\boldsymbol{I}_{i,j}$ as the pixel's brightness.

[8]Alpha, red, green, blue. Alpha is a coefficient for transparency of the color.

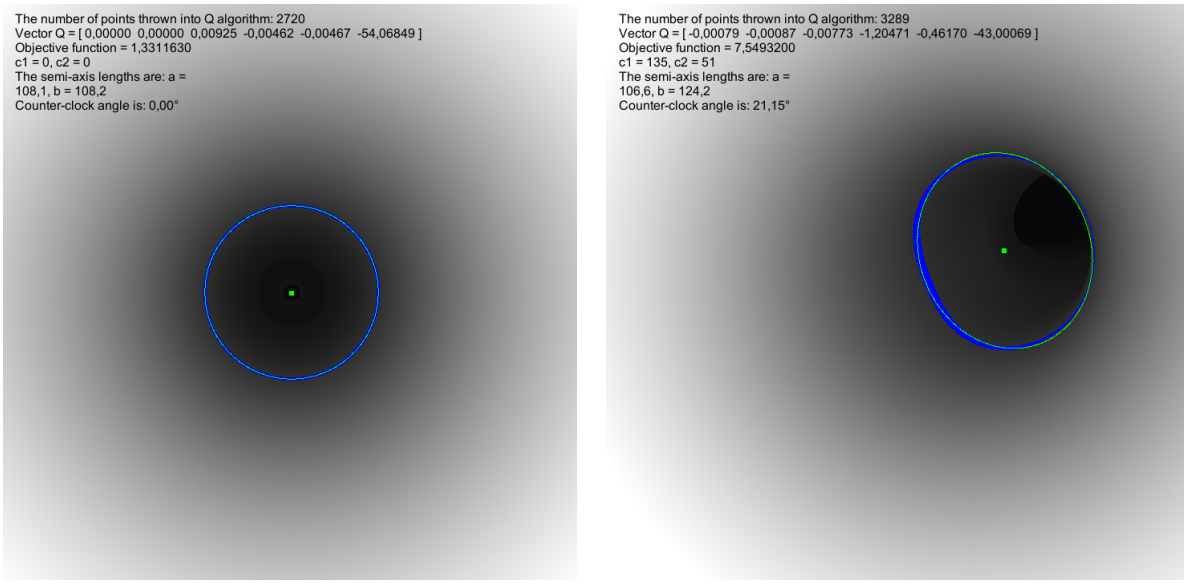


Figure 2.7: First algorithm result on Figures 2.1 and 2.2

number within zero and one. Now, let $\boldsymbol{H}_b$ be a $\boldsymbol{I}_{width} \times \boldsymbol{I}_{height}$ brightness highlighting matrix, where

$$\boldsymbol{H}_{b\,i,j} = \begin{cases} 1 & \text{if } \boldsymbol{I}_{i,j} \geq b_{min} \wedge \boldsymbol{I}_{i,j} \leq b_{max}, \\ 0 & \text{else,} \end{cases} \tag{2.13}$$

where b_{min} is the minimal brightness of the pixel and b_{max} is the maximal. For example, take Figures 2.1, 2.2 and set $b_{min} = 0.18$ and $b_{min} = 0.19$, $\boldsymbol{I}_{width} = \boldsymbol{I}_{height} = 720$. We will visualize the highlight matrix as blue pixels on the screen and draw green ellipse using the conic fitting algorithm (the method of implementing the algorithm is described at the end of this Subsection). For these values, the result is sufficient in case of the straight tube, but in case of the curved tube these points does not make convincing elliptic shape on their own as we can notice on Figure 2.7.

The second algorithm of highlighting points is based on the difference in brightness between neighborhoods. Let $\boldsymbol{H}_d$ be a $\boldsymbol{I}_{width} \times \boldsymbol{I}_{height}$ difference in brightness highlighting matrix, where

$$\boldsymbol{H}_{d\,i,j} = \begin{cases} 1 & \text{if } \max\{\boldsymbol{I}[i-1,i,i+1;j-1,j,j+1]\} - \\ & \quad - \min\{\boldsymbol{I}[i-1,i,i+1;j-1,j,j+1]\} > d, \\ 0 & \text{else,} \end{cases} \tag{2.14}$$

where d is the targeted difference in brightness. As with previous highlight matrix we take Figures 2.1, 2.2 and set $d = 0.03$. Unfortunately, Figure 2.1 is too smooth to find any points. From Figure 2.8 we ca observe, that the ellipse indicated in the contour, compared with the drawn ellipse, is not that precise. However, this filter can be used to extract exact ellipse parameters from Figures 2.5, 2.6. In the next Subsection we will work with pixels selection from the matrix $\boldsymbol{H}_b$.

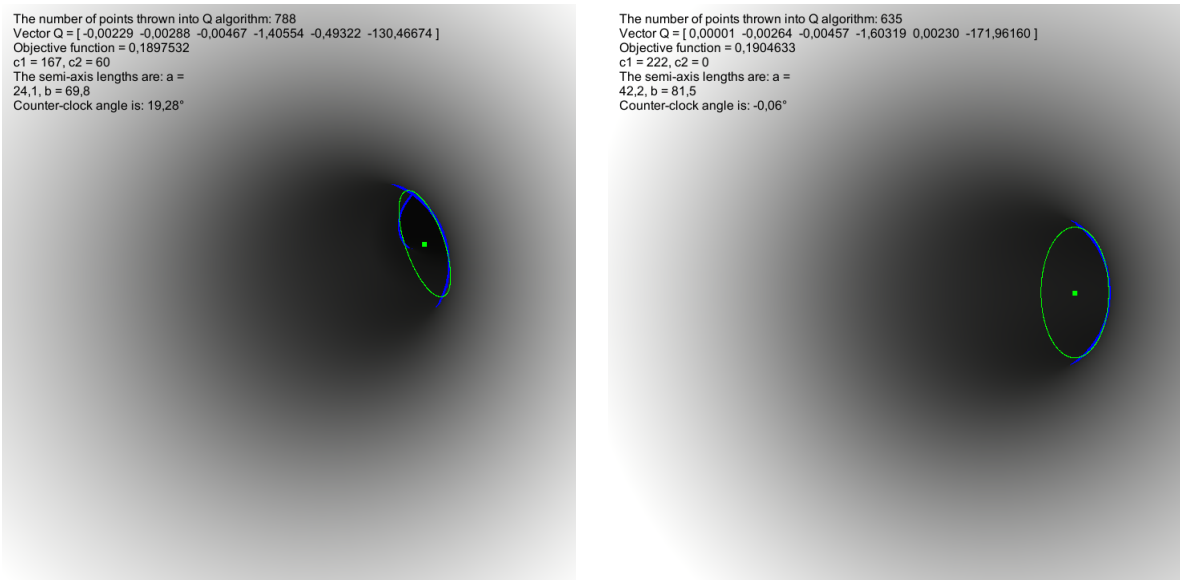


Figure 2.8: Second algorithm result on Figures 2.1 and 2.2

We start with the implementation of conic fitting algorithm (Subsection 1.2.5) by embedding points from the highlight matrix $\boldsymbol{H}$ to $\mathbb{G}_{5,3}$. Let $n = \boldsymbol{I}_{width}$ and $m = \boldsymbol{I}_{height}$, then the point set $\mathbb{U}$ is computed as follows,

$$\boldsymbol{u}_{i+n(j-1)} = \begin{cases} (i - \frac{1}{2}n)\boldsymbol{e}_1 + (j - \frac{1}{2}m)\boldsymbol{e}_2 \iff \boldsymbol{H}_{i,j} = 1, \\ \boldsymbol{o} \quad \text{else.} \end{cases}$$

Then we compute the matrix $\boldsymbol{P}$ (1.61),

$$\boldsymbol{P} = \sum_{i}^{nm} \boldsymbol{B}\mathcal{C}(\boldsymbol{u}_i)\mathcal{C}(\boldsymbol{u}_i)^T\boldsymbol{B}, \quad \boldsymbol{u}_i \in \mathbb{U}.$$

Rest of the steps are same as in Subsection 1.2.5.

2.3.3. Tube Axis Recovery

If we want to recover the tube axis using the above algorithms (resulting in a single ellipse), we have to find the connection between the shape and position of the ellipse on the screen and known environment properties (camera and tube properties). We can use projective geometry in computer graphics to see how the point in 3D is projected on the screen (to find more about computer graphics and projective geometry, see [4]).

In the computer graphics, camera has 3 main properties along with its position, rotation and resolution. The first property is a FOV (field of view) and the last two are z_{near}, z_{far}, i.e. the closest and the most distant point that can be projected. Now we consider that the camera has its own coordinate system (we consider that the camera is static in its own coordinate system and changes to its position and rotation are done in global coordinate system). The reason to do this is that we don't have to manipulate with the projection matrix $\boldsymbol{M}$ described below, but rather transform the rest of the space. The camera (in its own coordinate system) is placed in the origin $\boldsymbol{o} = (0, 0, 0)$ and it is pointed

to $\boldsymbol{e}_3$-axis $(0,0,1)$. Then the point $\boldsymbol{u} = (u_1, u_2, u_3) \in \mathbb{R}^3$ is projected using a projection matrix $\boldsymbol{M}$ (according to [8]),

$$
\begin{aligned}
\boldsymbol{M} &= \begin{pmatrix} \frac{\boldsymbol{I}_{height}}{\boldsymbol{I}_{width}} \cot(\frac{\mathrm{FOV}}{2}\frac{\pi}{180}) & 0 & 0 & 0 \\ 0 & \cot(\frac{\mathrm{FOV}}{2}\frac{\pi}{180}) & 0 & 0 \\ 0 & 0 & \frac{z_{far}}{z_{far}-z_{near}} & 1 \\ 0 & 0 & \frac{-z_{near}z_{far}}{z_{far}-z_{near}} & 0 \end{pmatrix}, \\
\boldsymbol{u}_{\boldsymbol{M}} &= \begin{pmatrix} u_1 & u_2 & p_3 & 1 \end{pmatrix}, \\
\overline{\boldsymbol{u}}_{proj} &= \boldsymbol{u}_{\boldsymbol{M}}\boldsymbol{M} = \\
&= \begin{pmatrix} u_1 \frac{\boldsymbol{I}_{height}}{\boldsymbol{I}_{width}} \cot(\frac{\mathrm{FOV}}{2}\frac{\pi}{180}) & u_2 \cot(\frac{\mathrm{FOV}}{2}\frac{\pi}{180}) & u_3 \frac{z_{far}}{z_{far}-z_{near}} + \frac{-z_{near}z_{far}}{z_{far}-z_{near}} & u_3 \end{pmatrix}, \\
\boldsymbol{u}_{proj} &= \begin{pmatrix} \frac{\boldsymbol{I}_{width}}{2}\frac{\overline{\boldsymbol{u}}_{proj}[1]}{\overline{\boldsymbol{u}}_{proj}[4]} & \frac{\boldsymbol{I}_{height}}{2}\frac{\overline{\boldsymbol{u}}_{proj}[2]}{\overline{\boldsymbol{u}}_{proj}[4]} \end{pmatrix} = \\
&= \frac{1}{2}\begin{pmatrix} \frac{u_1}{u_3}\boldsymbol{I}_{height} \cot(\frac{\mathrm{FOV}}{2}\frac{\pi}{180}) & \frac{u_2}{u_3}\boldsymbol{I}_{width} \cot(\frac{\mathrm{FOV}}{2}\frac{\pi}{180}) \end{pmatrix} = \\
&= \frac{1}{2u_3} \cot(\frac{\mathrm{FOV}}{2}\frac{\pi}{180}) \begin{pmatrix} u_1\boldsymbol{I}_{height} & u_2\boldsymbol{I}_{width} \end{pmatrix},
\end{aligned}
\tag{2.15}
$$

where $\overline{\boldsymbol{u}}_{proj}$ is a projection of the point $\boldsymbol{u}$ to a rectangle given by points $(-1,1,1)$,

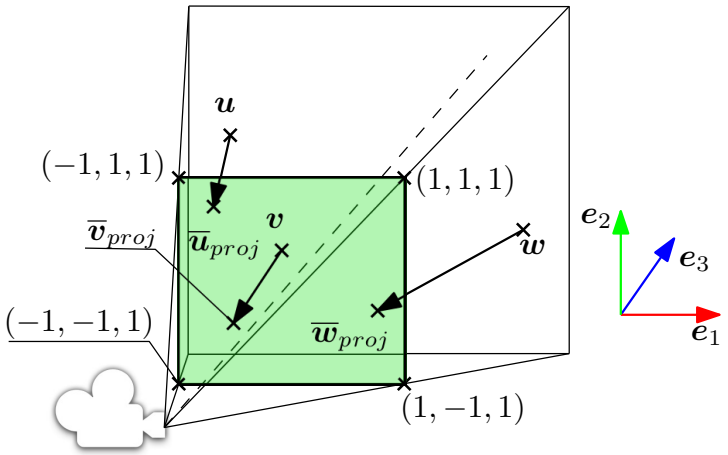


Figure 2.9: Projection of points into the rectangle

$(1,1,1)$, $(1,-1,1)$, $(-1,-1-1)$ as it is visualized in Figure 2.9. Term $\frac{\boldsymbol{I}_{width}}{\boldsymbol{I}_{height}}$ is commonly called an aspect ratio. Note that the point $\boldsymbol{u}$ is projected to the pixel with coordinates $\alpha(\boldsymbol{u})[1], \alpha(\boldsymbol{u})[2]$, where

$$
\alpha(\boldsymbol{u}) = \mathrm{Round}\begin{pmatrix} \boldsymbol{u}_{proj}[1] + \frac{1}{2}\boldsymbol{I}_{width} & \boldsymbol{u}_{proj}[2] + \frac{1}{2}\boldsymbol{I}_{height} \end{pmatrix}.
$$

Note that the following assumptions are made on the orthographic projection[9]. In the case when the generated ellipse is a circle, we can say that the $\boldsymbol{e}_3$ coordinate of the points on the circle is constant as we can see on Figure 2.10. In the case of an ellipse, when observing curved part of the tube, there exist 2 points with the same $\boldsymbol{e}_3$ coordinates, along

[9]Note that the views on Figures 2.10, 2.11 are not perspective, but for simplicity rather orthographic. As we will see, this approach

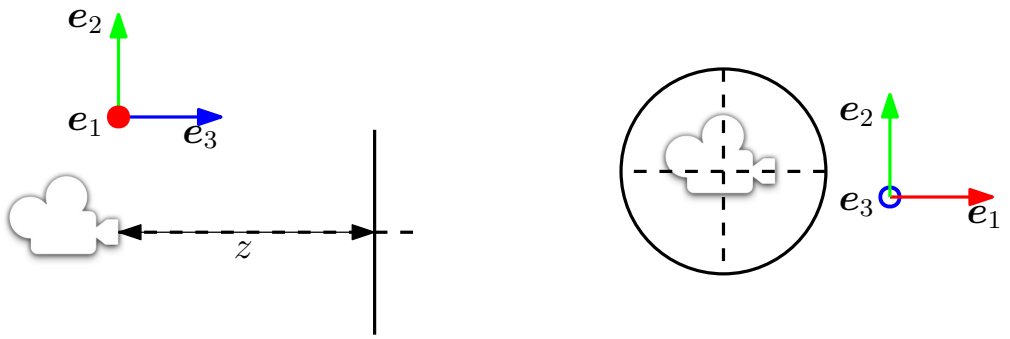


Figure 2.10: Orthographic view of a circle from distance

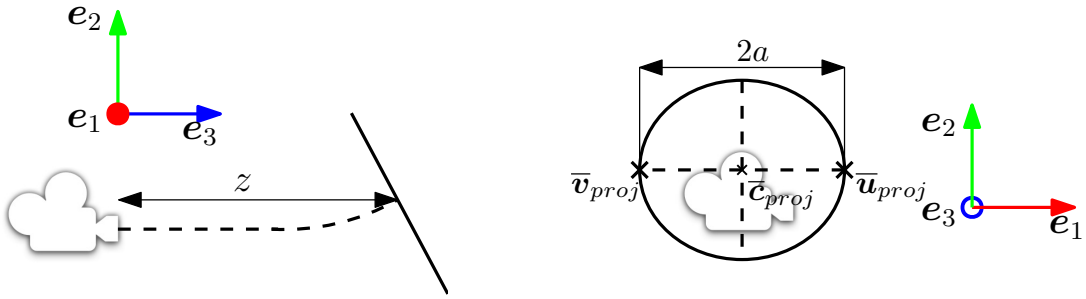


Figure 2.11: Orthographic view of rotated circle around $\boldsymbol{e}_1$ axis from distance

with ellipses center, as we can see in Figure 2.11. It is no surprise that these 2 points are on major semi-axis. If we compute distance of the points $\overline{\boldsymbol{u}}_{proj}$, $\overline{\boldsymbol{v}}_{proj}$, we get $2a$ (a is the length of ellipses major semi-axis).

$$\begin{aligned}
\overline{\boldsymbol{u}}_{proj} &= \begin{pmatrix} \overline{\boldsymbol{c}}_{proj}[1] + a & \overline{\boldsymbol{c}}_{proj}[2] \end{pmatrix} = \tfrac{1}{2u_3} \cot(\tfrac{\text{FOV}}{2} \tfrac{\pi}{180}) \begin{pmatrix} u_1 \boldsymbol{I}_{height} & u_2 \boldsymbol{I}_{width} \end{pmatrix}, \\
\overline{\boldsymbol{v}}_{proj} &= \begin{pmatrix} \overline{\boldsymbol{c}}_{proj}[1] - a & \overline{\boldsymbol{c}}_{proj}[2] \end{pmatrix} = \tfrac{1}{2v_3} \cot(\tfrac{\text{FOV}}{2} \tfrac{\pi}{180}) \begin{pmatrix} v_1 \boldsymbol{I}_{height} & v_2 \boldsymbol{I}_{width} \end{pmatrix},
\end{aligned}$$

where u_1, u_2, u_3 and v_1, v_2, v_3 are coefficients of 3D points $\boldsymbol{u}, \boldsymbol{v}$ and a is major semi-axis length. We can use that $u_3 = v_3$, $u_2 = v_2$ and calculate an Euclidean distance of these 2 points, which is $\|\cdot\|_2$ norm of vector $\boldsymbol{u} - \boldsymbol{v}$

$$\begin{aligned}
\|\boldsymbol{u} - \boldsymbol{v}\|_2 &= \sqrt{(\overline{\boldsymbol{c}}_{proj}[1] + a - \overline{\boldsymbol{c}}_{proj}[1] + a)^2 + (\overline{\boldsymbol{c}}_{proj}[2] - \overline{\boldsymbol{c}}_{proj}[2])^2} = 2a = \\
&= \tfrac{1}{2u_3} \cot(\tfrac{\text{FOV}}{2} \tfrac{\pi}{180}) \sqrt{(u_1 \boldsymbol{I}_{height} - v_1 \boldsymbol{I}_{height})^2 + (u_2 \boldsymbol{I}_{width} - v_2 \boldsymbol{I}_{width})^2} = \\
&= \tfrac{\boldsymbol{I}_{height}}{2u_3} \cot(\tfrac{\text{FOV}}{2} \tfrac{\pi}{180})(u_1 - v_1),
\end{aligned} \tag{2.16}$$

where $u_1 - v_1 = 2r$, where r is the radius and since u_3 is the only unknown in the previous equation we can compute it directly. Note that a is semi-axis length of the ellipse in perspective view, while r is actual tube radius. Thus

$$\begin{aligned}
2a &= \tfrac{\boldsymbol{I}_{height}}{2u_3} \cot(\tfrac{\text{FOV}}{2} \tfrac{\pi}{180}) 2r \\
c_3 = u_3 &= \tfrac{\boldsymbol{I}_{height}}{2} \cot(\tfrac{\text{FOV}}{2} \tfrac{\pi}{180}) \tfrac{r}{a}.
\end{aligned}$$

With all this in mind we are now able to compute the center's $\boldsymbol{e}_3$ coordinate of the conic. Now it is easy to compute the rest of the centers coordinates as follows

$$\begin{aligned}
\begin{pmatrix} \overline{\boldsymbol{c}}_{proj}[1] & \overline{\boldsymbol{c}}_{proj}[2] \end{pmatrix} &= \tfrac{1}{2c_3} \cot(\tfrac{\text{FOV}}{2} \tfrac{\pi}{180}) \begin{pmatrix} c_1 \boldsymbol{I}_{height} & c_2 \boldsymbol{I}_{width} \end{pmatrix} \iff \\
\iff \begin{pmatrix} c_1 & c_2 \end{pmatrix} &= 2c_3 \tan(\tfrac{\text{FOV}}{2} \tfrac{\pi}{180}) \begin{pmatrix} \overline{\boldsymbol{c}}_{proj}[1] \tfrac{1}{\boldsymbol{I}_{height}} & \overline{\boldsymbol{c}}_{proj}[2] \tfrac{1}{\boldsymbol{I}_{width}} \end{pmatrix}
\end{aligned}$$

The computer application has a camera with the following properties: FOV = 100, $z_{near} = 0.3$, $z_{far} = 1000$ and $\boldsymbol{I}_{height} = \boldsymbol{I}_{width} = 720$. Then we approximate the ellipses center $\boldsymbol{c} = (c_1, c_2, c_3) \in \mathbb{R}^3$ in camera coordinate system by

$$\begin{aligned} c_3 &= 360 \cot(\tfrac{5\pi}{18}) \tfrac{r}{a}, \\ c_1 &= \tfrac{c_3}{360} \tan(\tfrac{5\pi}{18}) \overline{\boldsymbol{c}}_{proj}[1] = \tfrac{r}{a} \overline{\boldsymbol{c}}_{proj}[1], \\ c_2 &= \tfrac{c_3}{360} \tan(\tfrac{5\pi}{18}) \overline{\boldsymbol{c}}_{proj}[2] = \tfrac{r}{a} \overline{\boldsymbol{c}}_{proj}[2], \end{aligned} \tag{2.17}$$

where $\overline{\boldsymbol{c}}_{proj}[1], \overline{\boldsymbol{c}}_{proj}[2] \in \mathbb{R}^2$ is center of the ellipse computed by conic fitting algorithm, a is its major semi-axis length, and r is radius of the tube. Remember that this center $\boldsymbol{c}$ is related to the camera and we have to place it into global coordinate system, which can be done using Euler angles of the camera rotation. Euler angles are commonly used in computer graphics and they are easy to work with. Recall from linear algebra that rotation can be computed using rotation matrix for each axis

$$\begin{aligned} \boldsymbol{R}_{\boldsymbol{e}_1}(\theta) &= \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos(\theta) & \sin(\theta) \\ 0 & -\sin(\theta) & \cos(\theta) \end{pmatrix}, \\ \boldsymbol{R}_{\boldsymbol{e}_2}(\theta) &= \begin{pmatrix} \cos(\theta) & 0 & -\sin(\theta) \\ 0 & 1 & 0 \\ \sin(\theta) & 0 & \cos(\theta) \end{pmatrix}, \\ \boldsymbol{R}_{\boldsymbol{e}_3}(\theta) &= \begin{pmatrix} \cos(\theta) & \sin(\theta) & 0 \\ -\sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{pmatrix}. \end{aligned}$$

Let $\theta_{\boldsymbol{e}_1}, \theta_{\boldsymbol{e}_2}, \theta_{\boldsymbol{e}_3}$ be Euler angles of camera rotation in global coordinate system. Let $\boldsymbol{w} = (w_1, w_2, w_3) \in \mathbb{R}^3$ be a camera position in global coordinates, then the center of the ellipse $\boldsymbol{c}$ is transformed to global coordinate system center $\hat{\boldsymbol{c}}$ as follows

$$\hat{\boldsymbol{c}} = \boldsymbol{w} + \boldsymbol{c}\boldsymbol{R}_{\boldsymbol{e}_1}(\theta_{\boldsymbol{e}_1})\boldsymbol{R}_{\boldsymbol{e}_2}(\theta_{\boldsymbol{e}_2})\boldsymbol{R}_{\boldsymbol{e}_3}(\theta_{\boldsymbol{e}_3}). \tag{2.18}$$

Now we estimate the tube axis. Suppose, that the camera is moving along an exact axis trough a part of the tube with the following restrictions. The look rotation of the camera goes with a tangent of the axis at a point where the camera is placed. Every step $i \in \{1, \ldots, n\}$ camera captures an image, from which we will obtain highlight matrix. Using conic fitting algorithm, we compute the ellipse properties. From that properties and the camera properties, we can compute the i-th point of estimated tube. Then the tube sector is divided into n points and the estimate of the axis is then set of points $\hat{\boldsymbol{c}}_i \in \mathbb{R}^3, i = 1, \ldots, n$

$$\begin{aligned} \hat{\boldsymbol{c}}_1 &= \boldsymbol{w}_1 \\ \hat{\boldsymbol{c}}_i &= \boldsymbol{w}_i + \boldsymbol{c}_i \boldsymbol{R}_{i,\boldsymbol{e}_1}(\theta_{i,\boldsymbol{e}_1}) \boldsymbol{R}_{i,\boldsymbol{e}_2}(\theta_{i,\boldsymbol{e}_2}) \boldsymbol{R}_{i,\boldsymbol{e}_3}(\theta_{i,\boldsymbol{e}_3}), \end{aligned} \tag{2.19}$$

where $\boldsymbol{w}_i$ is position of the camera in global coordinate system, $\boldsymbol{c}_i$ is position of center of the ellipse in camera coordinate system and $Rt_{i,\boldsymbol{e}_1}(\theta_{i,\boldsymbol{e}_1})\boldsymbol{R}_{i,\boldsymbol{e}_2}(\theta_{i,\boldsymbol{e}_2})\boldsymbol{R}_{i,\boldsymbol{e}_3}(\theta_{i,\boldsymbol{e}_3})$ are Euler angles from global coordinate system to cameras.

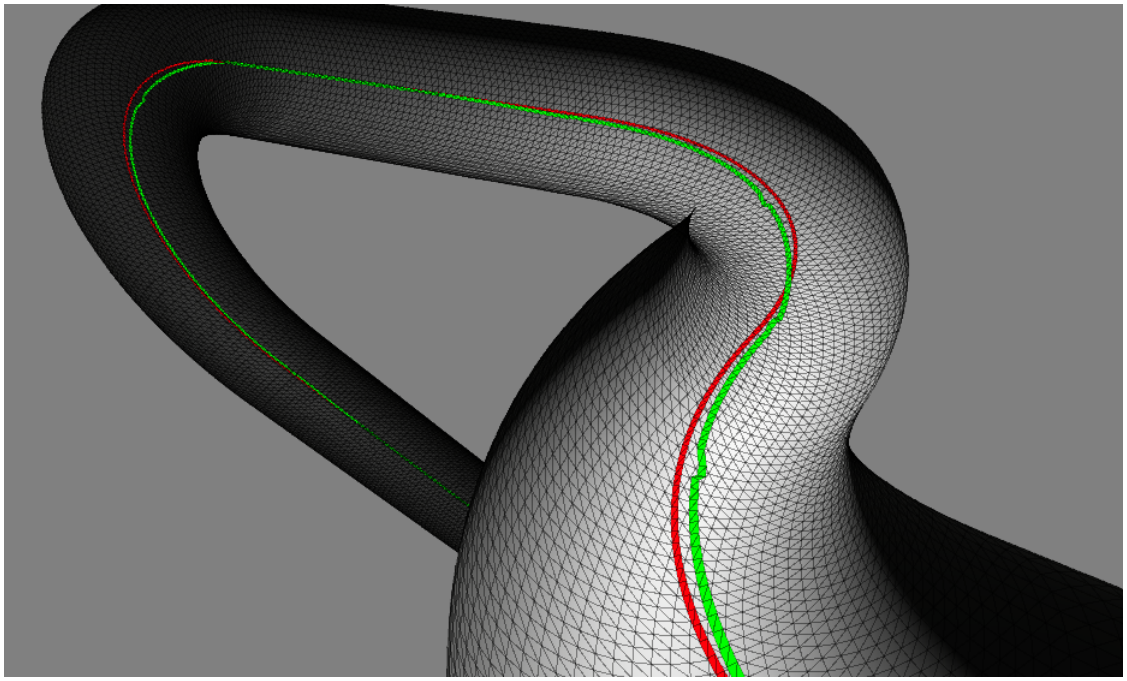


Figure 2.12: Perfect scenario simulation, where the exact elliptic projection of circle is obtained from intersection of plane (distance from camera is $\frac{8}{3}r$) with the tube.

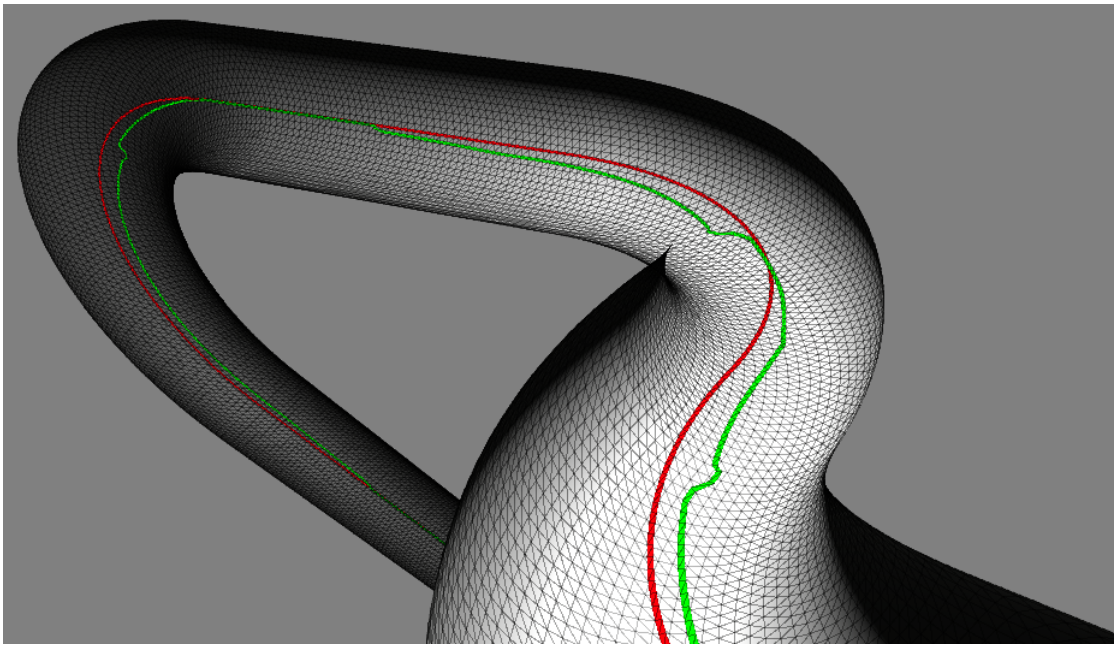


Figure 2.13: The axis is computed using $\boldsymbol{H}_b$ (2.13), with values $b_{min} = 0.18, \quad b_{max} = 0.19$.

Note that even the perfect scenario showed in Figure 2.12 does not give exact axis back. This is due to perspective projection properties, which are noticeable from Figure 2.14.

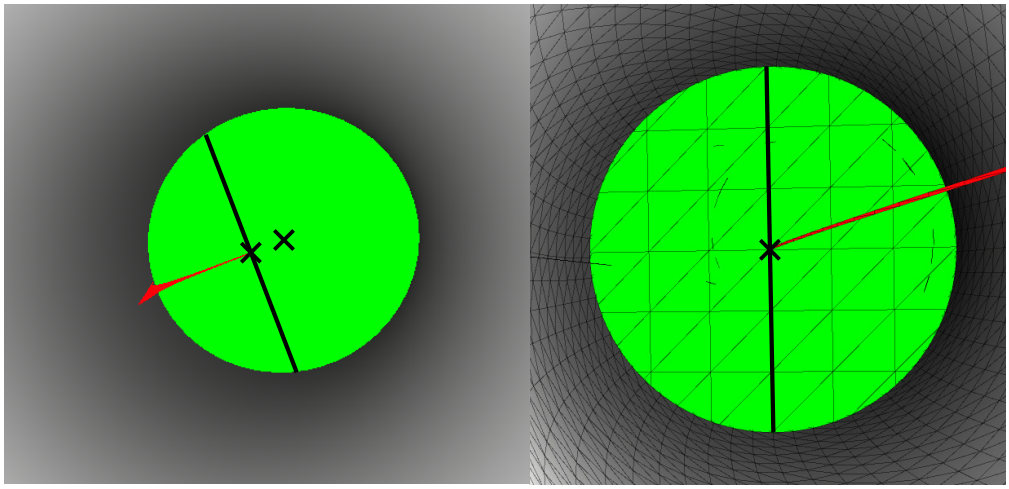
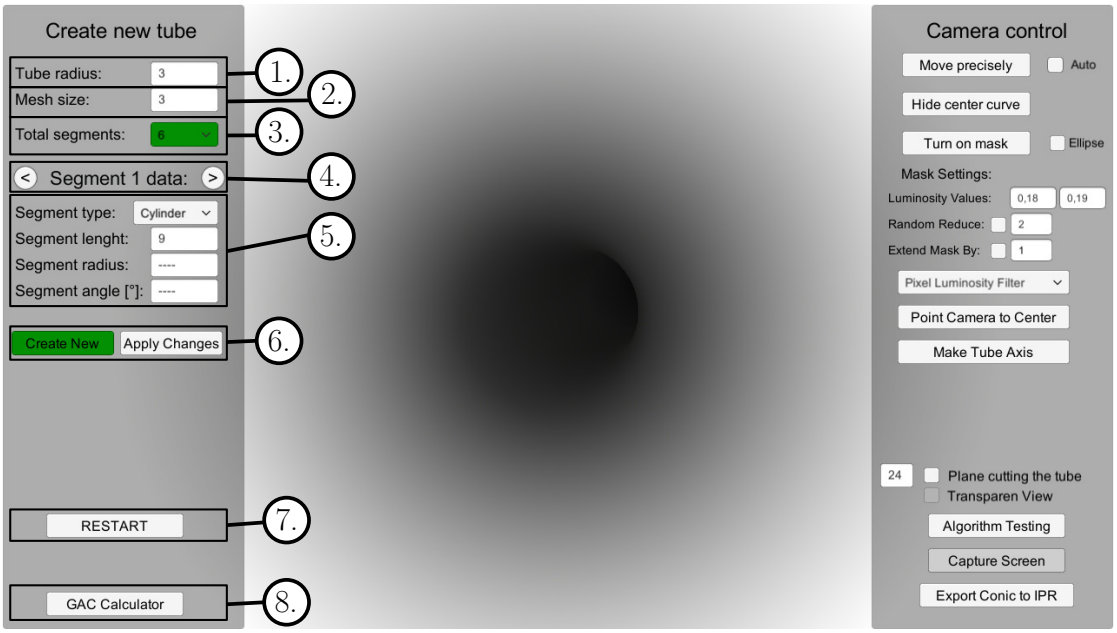


Figure 2.14: Inaccuracy in perspective projection.

2.3.4. Application documentation

This Subsection is guide to the application interface, with a brief explanation of each function. First Figure is for tube generation. Note that all changes are done after pressing 'Apply Changes'.



1. Change tube radius by rewriting the number in input field.

2. Change mesh size by rewriting the number in input field. This number is used to calculate m, the number of points in one circle as you can see in Figure 2.3. $m = \lceil 2\pi \cdot r \cdot \text{meshsize} \rceil$, where $\lceil \rceil$ is ceiling function.

3. Choose the number of tube segments in dropdown. You can choose from 1 up to 10 total segments.

4. In this part of the menu, you can navigate through each of the segments.

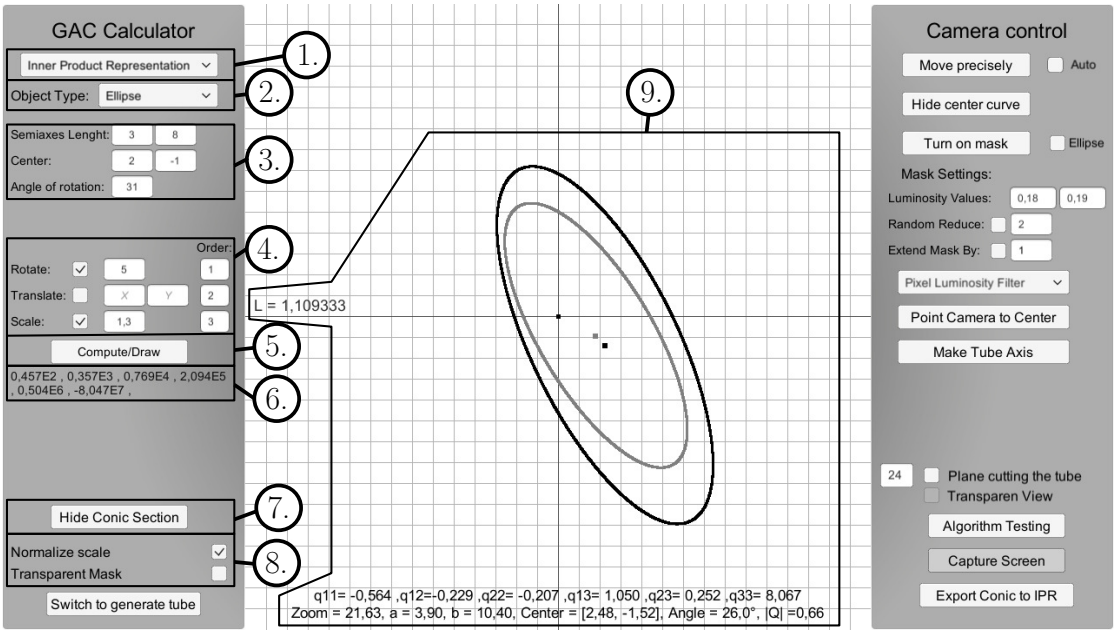
5. In this part you can set data to each segment. 'Segment type' can be either 'Cylinder' or 'Torus'. To each of the 'Segment type' you can assign 'Segment length'. If the 'Torus' is selected, you can also assign 'Segment radius', which affect the radius of a circle part of the tube segment axis. And with selecting 'Torus', you can also select 'Segment angle', where with 0° the tube will continue down, with 90° the tube will continue right and so on.

6. If you hit button 'Create New', program will create n segments with default settings ('Segment type' = 'Cylinder', 'Segment length' = '1'). If you hit 'Apply Changes' button, program will apply all changes you made excluding changing number of segments.

7. Push this button to set tube data to its default values.

8. Push this button to change interface to GAC Calculator.

Second Figure describe infertace for GAC Calculator.



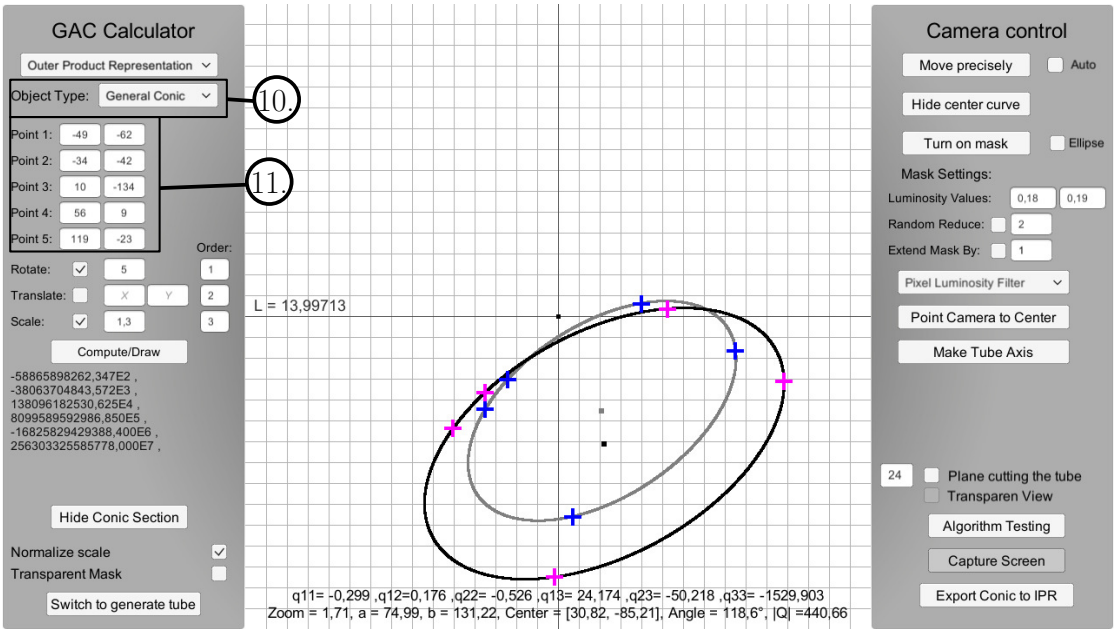
1. Choose either 'Inner Product Representation' or 'Outer Product Representation' in this dropdown. Note that the input interface depends on the item selected 3. for first item and 12. for second.

2. Choose either 'Ellipse' or 'Hyperbola'.

3. This area is for input values for the conic in inner product representation.

4. In this area you can set transformations. You can also choose an order of performing the transformation (by choice of numbers 1,2,3).

5. Push the button to compute a multivector in $\mathbb{G}_{5,3}$ basis. Program will compute conic before and after the transformations are done. The original conic is gray and the transformed is black.



6. In this area, you can find an information about multivector in the form $a_2\boldsymbol{E}_2$, $a_3\boldsymbol{E}_3$, $a_4\boldsymbol{E}_4$, $a_5\boldsymbol{E}_5$, $a_6\boldsymbol{E}_6$, $a_7\boldsymbol{E}_7$, where $\boldsymbol{E}_2 = \overline{\boldsymbol{n}}_+$, $\boldsymbol{E}_3 = \overline{\boldsymbol{n}}_-$, $\boldsymbol{E}_4 = \overline{\boldsymbol{n}}_\times$, $\boldsymbol{E}_5 = \boldsymbol{e}_1$, $\boldsymbol{E}_6 = \boldsymbol{e}_2$, $\boldsymbol{E}_7 = \boldsymbol{n}_+$.

7. If press this button to hide/show an Image field (a place where you can see graphical output).

8. If you check a 'Normalize Scale' toggle, it adapts screen to the computed conic (conic has preset size on the screen). Default resolution is 720×720, where the origin is placed in the middle of screen. If you check a 'Transparent Mask' toggle, screen will be transparent and you will be able to see a view inside the tube.

9. This area contains the resulting conic, an information about a scale of one square L and conic properties derived from the general equation.

10. In this dropdown you can choose either 'General Conic', 'Axes-Aligned Conic' or 'Circle'. Changing item selected from the dropdown will change a number of accessible points (5 for the first item, 4 for the second item, and 3 for the third item).

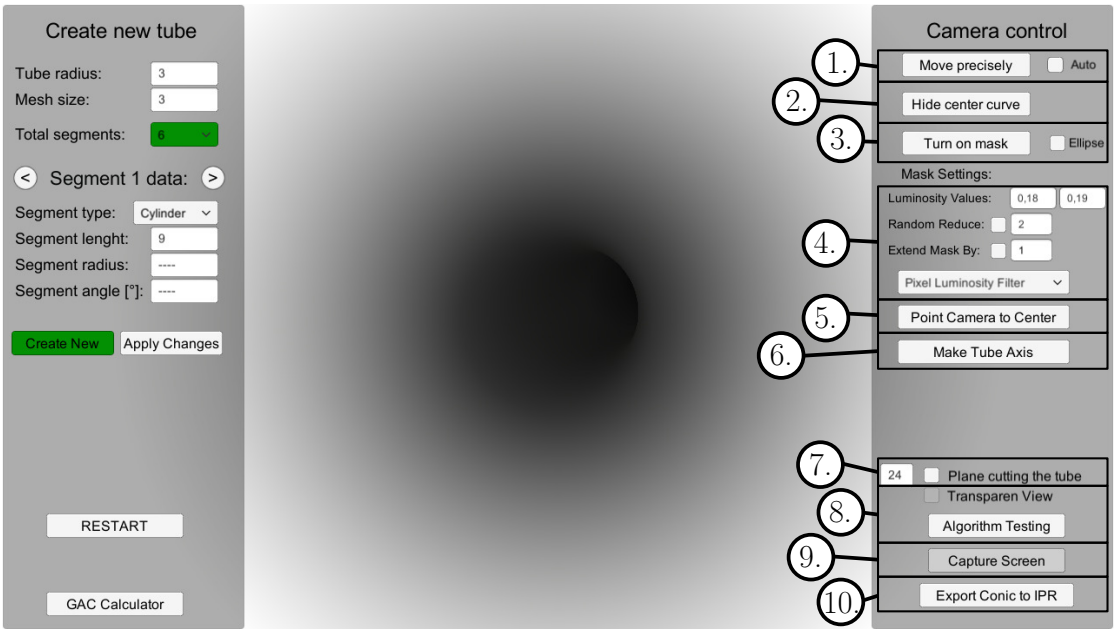
11. In this area you can set points to span the conic by number of points depending on dropdown 10. item selection.

Last Figure is used to describe functions connected to the conic fitting algorithm.

1. Push this button to translation of the camera and its light forward by one step. In addition if a toggle 'Auto' is checked, the camera will attempt to move 10 times per second.

2. Push the button to Hide/show center curve (tube axis).

3. Push the button to activate/deactivate creation of highlight matrix depends on item selection in dropdown in 4. In addition if a toggle 'Ellipse' is checked, a conic fitting algorithm will draw closest conic to the set of points (at least 5 points are required to perform).



4. Choose item in dropdown and set properties of the filter (mask). The first is 'Pixel Luminosity Filter'; 'Luminosity Values' property controls certain pixel brightness values (Matrix $\boldsymbol{H}_b$). The second is 'Difference in Luminosity' and it computes minimal and maximal brightness of surrounding pixels (Matrix $\boldsymbol{H}_d$). A 'Random Reduce' property causes that only one of n highlighted pixels is displayed and counted. A 'Extend Mask By' counts not only the highlighted pixel, but even n pixels surrounding the pixel.

5. Push this button to simulate moving through the tube, during the simulation interface is not interactable. If you select in dropdown 4. the first item, the simulation is performed only with matrix $\boldsymbol{H}_b$. If the second item is selected, to perform the perfect scenario.

6. Push this button to point camera to the point on the center axis, which is '24' steps ahead. You can change this value by changing the number in input field 7.

7. In this area if you check a 'Plane cutting the tube' it shows a plane in distance of a number written in input field.

8. Push the button to create a field where you can click with mouse. On that place it creates a point and if there are at least 5 points, the conic fitting algorithm will draw a conic closest to these points.

9. Push this button to capture and save screenshot to folder 'RenderOutput' in png format.

10. Push this button to export conic data to inner product representation input 3. Input field has to be active to perform export.

Conclusion

This thesis aimed to a tube axis estimation, which consists of projected ellipse centers into environment of the tube. Chapter 1 gave us necessary mathematical background. We defined a relation between the position of the basis blade in its basis and its signature (1.12). This relation helped with the definition of multivector in *C#*, more specifically with the grade projection, and actual transition between the form of the basis blades ($\boldsymbol{e}_1 \longleftrightarrow \boldsymbol{E}_2$). We used a fact that a multivector can be written as a linear combination of basis blades to define that geometric, inner and outer products can be computed by (1.14), (1.20), (1.21). We introduced inner (outer) product null space of a blade as a set of vectors, whose inner (outer) product with the blade is zero. Important fast was that IPNS and OPNS representations are related by duality. In Section 1.2 we defined transition between the standard basis of $\mathbb{R}^{5,3}$ to basis $\mathbb{R}^{5,3^*}$, with corresponding bilinear form $\boldsymbol{B}$ (1.31). Figures with examples were presented with data proving that the code was precisely implemented. We have to point out that the translation of a conic in directions $\boldsymbol{e}_1$, $\boldsymbol{e}_2$ (1.54), (1.55) are giving non-zero coefficients for the basis blades $\boldsymbol{n}_-$, $\boldsymbol{n}_+$. However, it has no impact on software functionality because, as equation (1.40) showed, basis vectors $\boldsymbol{n}_-$ and $\boldsymbol{n}_+$ are orthogonal to all embedded points, so the conic remains the same regardless the coefficients (in other words, equation (1.38) remains the same regardless $v_\times$,v_-). In Subsection 1.2.5 we explained conic fitting algorithm introduced in [6]. We have shown that one point can change the result dramatically (Figures 1.10,1.11).

In Section 2.1 we implemented necessary algorithms to be able to perform geometric, inner and outer product. Then we implemented efficient method for computing geometric product of two basis blades as a basis blade with position from the value matrix (Code 2.9) with precomputed (Code 2.7 and Code 2.8) values for the geometric product. An improvement can be made by generalizing this concept to the general geometric algebra $\mathbb{G}_{p,q}$ and move the computation of the value matrix to C# . In Section 2.2 we defined two functions ϕ (2.2) and ϕ^{-1} (2.3), which transit vectors between the bases $\overline{\mathbb{R}}^{5,3}$ and $\overline{\mathbb{R}}^{5,3^*}$. Then, using ϕ^{-1}, we implemented inner and outer product null space representations of conics to C# code. Examples in Section 1.2 verified that this code was successfully implemented and introduced software's interface for creating conics. In Section 2.3 we created 3D environment including light source, camera and tube. From image data (Figures 2.1 and 2.2) we were able to create algorithms for choosing certain points and extract an ellipse. However, only the first algorithm (using matrix $\boldsymbol{H}_b$) was able to estimate the tube axis, that was actually going through the tube (Figure 2.13). We compared that algorithm to the best scenario (Figure 2.12), which took advantage of the cutting plane (Figures 2.5 and 2.6). The inaccuracy, even in the best scenario, is caused by projective geometry in computer graphics, where the size of the projected object depends only on $\boldsymbol{e}_3$ coordinate. This causes that the distant objects (in $\boldsymbol{e}_1, \boldsymbol{e}_2$ coordinates) seems closer than they really are. The next step would be an image processing to reduce the inaccuracy, or even compute the inaccuracy from the shape of the ellipse. Even considerable are self-learning algorithms that can choose points based on different tube examples and textures. The result given by matrix $\boldsymbol{H}_b$ is sufficient for demonstration of approach to autonomous navigation of a robot with camera in the tube.

Bibliography

[1] Artin E.: *Affine and Projective Geometry*, 1988 [online], [cit. 2021-05-05].
URL <https://doi.org/10.1002/9781118164518.ch2>

[2] Ayoub B. Ayoub: *The Central Conic Sections Revisited*, Taylor & Francis , 1993.[online], [cit.2021-04-28], pages 322-325.
URL <https://doi.org/10.1080/0025570X.1993.11996157>

[3] Dorst, L., Fontijne, D., Mann, S.: *Geometric Algebra for Computer Science (Revised Edition)*. Morgan Kaufmann Publishers, 2007. ISBN 978-0-12-374942-0.

[4] Herman, I., Editors: Hewitt, W. T., Grave, M., Roch, M. *Projective Geometry and Computer Graphics*, Berlin: Springer Heidelberg, 1991. ISBN 978-3-642-84060-9. pages 28-61

[5] Hildenbrand, D.: *Foundantions of Geometric Algebra Computing*. Berlin: Springer Heidelberg, 2013. ISBN 978-3-642-31794-1.

[6] Hrdina, J., Návrat, A. & Vašík, P.: *Conic Fitting in Geometric Algebra Setting. Adv. Appl. Clifford Algebras 29, 72*, 2019 [online], [cit. 2021-05-01].
URL <https://doi.org/10.1007/s00006-019-0989-5>

[7] Hrdina, J., Návrat, A. & Vašík, P.: *Geometric Algebra for Conics. Adv. Appl. Clifford Algebras*, 2018 [online], [cit. 2021-05-01].
URL <https://doi.org/10.1007/s00006-018-0879-2>

[8] *3D Graphics with OpenGL*, Internet site. [online], [cit. 2021-05-05].
URL <https://www3.ntu.edu.sg/home/ehchua/programming/opengl/CG_BasicsTheory.html>

[9] Machálek, L.: *Korekce obrazových vad pomocí CGA* [online], [cit. 2021-04-08].
URL <https://www.vutbr.cz/studenti/zav-prace/detail/109029>

[10] Perwass, C.: *Geometric algebra with applications in engineering*. Berlin: Springer, c2009. ISBN 354089067X.

[11] Richter-Gebert, Jrgen: *Perspectives on Projective Geometry*. Springer Publishing Company, Incorporated, 2011. ISBN 978-3-642-17285-4.

[12] Smith, C.: *On Vertex-Vertex Systems and Their Use in Geometric and Biological Modelling*, University of Calgary, 2006. ISBN 9780494195741.

[13] Solomon, C.J.; Breckon, T.P.: *Fundamentals of Digital Image Processing: A Practical Approach with Examples in Matlab*, Wiley-Blackwell, c2010. ISBN 978-0470844731.

[14] Young, C. Y.: *Precalculus; Chapter 9*. John Wiley and Sons, c2010. ISBN 978-0-471-75684-2.

Appendices

A *C# _code*, folder including C# source code for Unity engine

B *Matlab_code*, folder including MATLAB source code

C *Version 1.0*, folder including Unity files, screenshots are saved to *RenderOutput*

D *GACImageProcessing.exe*