FAKULTA
INFORMAČNÍCH
TECHNOLOGIÍ
ČVUT V PRAZE

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Název:	Reprezentace vektorových prostorů ve virtuální realitě
Student:	Tomáš Bašta
Vedoucí:	doc. Ing. Mgr. Petr Klán, CSc.
Studijní program:	Informatika
Studijní obor:	Webové a softwarové inženýrství
Katedra:	Katedra softwarového inženýrství
Platnost zadání:	Do konce letního semestru 2020/21

Pokyny pro vypracování

Cílem práce je vytvoření, modelování a vizuální programování virtuálního světa reprezentujícího vektorové prostory. Virtuální vektorový prostor bude 3D a bude zahrnovat operace s vektory a jejich vlastnosti, lineární transformace vektorů a jejich vlastnosti a ortogonalitu vektorů včetně příslušných operací s ortogonálními vektory. Předpokládejte jeho edukativní využití.

Při práci postupujte následujícím způsobem:

1. Analyzujte vektorové prostory a jejich vlastnosti.
2. Analyzujte vybrané maticové transformace vektorů.
3. Analyzujte ortogonalitu vektorů včetně vlastností a základních operací.
4. Seznamte se se systémem NeoS VR a vizuálním programováním virtuálních scén LogiX.
5. Vytvořte nový virtuální svět, příslušně upravte jeho scénu a vytvořte základní prvky scény.
6. Vizuálně naprogramujte reprezentaci vektorového prostoru, maticové transformace a ortogonalitu včetně jejich vlastností.
7. Virtuální svět zveřejněte ve virtuálním hubu.

Seznam odborné literatury

Dodá vedoucí práce.

Ing. Michal Valenta, Ph.D.	doc. RNDr. Ing. Marcel Jiřina, Ph.D.
vedoucí katedry	děkan

V Praze dne 5. listopadu 2019

FACULTY
OF INFORMATION
TECHNOLOGY
CTU IN PRAGUE

Bachelor's thesis

Representation of Vector Spaces in Virtual Reality

Tomáš Bašta

Department of Software Engineering
Supervisor: doc. Ing. Mgr. Petr Klán, CSc.

June 1, 2020

Acknowledgements

I would like to express my deepest gratitude to my supervisor, doc. Ing. Mgr. Petr Klán, CSc., for his great enthusiasm for my thesis and for always making the time for me. Next, I would like to thank my girlfriend for reading every word of this thesis and her invaluable input. Lastly, I would like to thank my parents for all the support and encouragement they have given me during my studies.

Declaration

Citation of this thesis

Bašta, Tomáš. *Representation of Vector Spaces in Virtual Reality.* Bachelor's thesis. Czech Technical University in Prague, Faculty of Information Technology, 2020.

Abstract

This thesis describes the process of designing and implementing a world in virtual reality that represents vector spaces. The theoretical part describes vector spaces and introduces the concepts of virtual reality along with the NeosVR application, which is used to create the virtual world. In the practical part, the world is designed and implemented in NeosVR using visual programming language LogiX. The result of this work is an original and fully functional virtual world placed in the community hub of NeosVR. This world allows everyone to visualize fundamental concepts of vector spaces and can be therefore used as a supporting tool for classes of linear algebra.

Keywords design, implementation, virtual world, virtual reality, visualization, vector spaces, vector algebra, NeosVR, LogiX

Abstrakt

Tato práce popisuje proces návrhu a implementace světa ve virtuální realitě, který reprezentuje vektorové prostory. Teoretická část práce popisuje vektorové prostory a uvádí koncepty virtuální reality společně s aplikací NeosVR, která je použita k vytvoření virtuálního světa. V praktické části je svět navržen a naimplementován v aplikaci NeosVR za pomoci vizuálního programovacího jazyka LogiX. Výsledkem této práce je originální a plně funkční virtuální svět, který je umístěn v komunitním centru aplikace NeosVR. Tento svět umožňuje každému vizualizovat základní koncepty vektorových prostorů a může tedy sloužit jako pomůcka pro výuku linearní algebry.

Klíčová slova návrh, implementace, virtuální svět, virtuální realita, vizualizace, vektorové prostory, vektorová algebra, NeosVR, LogiX

Contents

Introduction **1**

1 Objectives **3**

2 Vector spaces **5**
- 2.1 Fields . . . 5
- 2.2 Vector spaces . . . 6
- 2.3 Properties . . . 8
- 2.4 Vector algebra . . . 9
  - 2.4.1 Vector equality . . . 9
  - 2.4.2 Scalar multiplication . . . 10
  - 2.4.3 Vector addition and subtraction . . . 11
  - 2.4.4 Dot product . . . 12
  - 2.4.5 Cross product . . . 13
- 2.5 Matrix transformations . . . 14
  - 2.5.1 Scaling . . . 15
  - 2.5.2 Shearing . . . 16
  - 2.5.3 Rotation . . . 16
- 2.6 Linear dependence . . . 17
- 2.7 Bases . . . 19
  - 2.7.1 Cramer's rule . . . 20
- 2.8 Orthogonality . . . 21

3 Virtual reality **23**
- 3.1 Uses of virtual reality . . . 24
- 3.2 Developing for virtual reality . . . 25
- 3.3 NeosVR . . . 26
  - 3.3.1 Creating new worlds . . . 27
  - 3.3.2 Filling up the worlds . . . 28
  - 3.3.3 The Inspector . . . 28

3.3.4 LogiX 31

4 Designing the world 35

4.1 Environment 35
4.2 Vector space 35
4.3 Vector tool 36
4.4 Interactive board 36

5 Implementing the world 39

5.1 Environment 39
5.2 Vector space 39
5.2.1 Points 39
5.2.2 Vectors 41
5.3 Vector tool 42
5.4 Interactive Board 44
5.4.1 Examples of use 46
5.5 Finishing touches 48

Conclusion 49

Bibliography 51

A Acronyms 53

B Contents of enclosed CD 55

List of Figures

2.1 Representation of vector in $\mathcal{R}^2$ (a) and $\mathcal{R}^3$ (b) 8
2.2 Visual representation of vector properties 9
2.3 Equal vectors 10
2.4 Visualization of scalar multiplication 11
2.5 Representation of x (black), y (black) and $x + y$ (red) 11
2.6 Geometrical significance of cross product [5] 13
2.7 Scaling transformation where $\alpha = 2$ and $\beta = -0.5$ 16
2.8 Shearing transformation parallel to the x-axis where $\lambda = 2$ 16
2.9 Rotation transformation where $\varphi = 60°$ 17
2.10 Set of linearly dependent vectors x (black), y (black) and z (red) . 19

3.1 Virtual reality headset with two controllers (Oculus Rift) [11] . . . 23
3.2 Fishing game in virtual reality (Real VR Fishing) [12] 24
3.3 Dissection of a frog in virtual reality (VictoryXR) [14] 24
3.4 Developing for VR in Unity [18] 25
3.5 Belt menu inside of NeosVR 26
3.6 Content hub allowing to browse and enter worlds 27
3.7 New world configuration 27
3.8 The inventory showcasing tools in the public section 28
3.9 The inspector for the Example slot with two components 29
3.10 The Human Body hierarchy 30
3.11 LogiX nodes menu 31
3.12 LogiX program composed of two int inputs, addition, and display . 32
3.13 LogiX controlled text object 33
3.14 LogiX program computing dot product organized via Blueprint . . 34

4.1 Wireframe of the Vector algebra tab 37

5.1 LogiX blueprint transforming position into a coordinates label . . 40
5.2 The point representation 40
5.3 LogiX blueprint setting the vector's anchors 41

5.4 The vector representation 42
5.5 LogiX blueprint switching the tool's modes 42
5.6 BPMN diagram of Vector Tool flow 43
5.7 LogiX blueprint creating a new vector 43
5.8 Using Vector tool to select vector 44
5.9 Interactive board with the vector algebra tab selected 45
5.10 Different geometrical representations of vector addition 46
5.11 Calculating a dot product of two vectors 46
5.12 Rotating vector around the y-axis by 90° 47
5.13 Visualization of vector rotation around the y-axis by 90° 47
5.14 Expressing vector by coordinates relative to basis X 48
5.15 Scene showcasing the finished world 48

Introduction

Math is, without a doubt, one of the greatest inventions of human civilization. It allows us to understand the fundamental rules of the world around us; however, some of the concepts and areas; are harder to understand than others. One such area is algebra, more specifically linear algebra, which introduces concepts like vectors and vector spaces. These concepts are fundamentally simple, but get more difficult to comprehend (and represent) with increasing dimensions.

The issue is; that without proper representation, it is difficult for students (or anyone else looking at it for the first time) to grasp. Currently, there are a lot of visualization tools available, allowing students to work with vectors and vector spaces in 2 dimensions. There are also a few other tools, which choose to represent three-dimensional vector spaces in 2D, but because of the simplification, they aren't as useful as one would think.

That is, where the idea for this thesis came from, to truly represent three-dimensional vector spaces in a virtual world, which will be accomplished by virtual reality. Thanks to that, the user will be transported into the virtual world of vectors and vector spaces. Here, the user will be able to create and operate with vectors, perform various vector operations, visually observe its results, and, hopefully, understand what these theoretical concepts mean in practice.

In the theoretical part, the theory of vector spaces is explained, which will provide a mathematical foundation for the world. Next, the NeosVR application and the visual programming language LogiX are introduced, which will be used to create the virtual world.

In the practical part, the virtual world is designed and implemented in NeosVR by utilizing the knowledge obtained in the theoretical part of the thesis.

Because this thesis is the first of such kind, it does not expand on any previously done work, and all the solutions presented here are original and completely new.

Chapter 1

Objectives

The main objective of this thesis is to create, model, and visually program a three-dimensional virtual world in NeosVR application, which will represent vector spaces. The world will also be added into a community hub, allowing other users to use it for educational purposes. This work will concretely cover the following steps:

Study of used concepts from linear algebra.
This chapter explains the needed concepts from linear algebra. Namely: vector spaces and its properties, selected matrix transformations of vectors and vector's orthogonality including operations with orthogonal vectors.

Introduction to virtual reality and NeosVR.
This chapter introduces the fundamental terms of virtual reality, its use in today's world, and educational benefits. Part of this chapter also presents the NeosVR application and explores its core concepts.

Designing the virtual world.
This chapter presents the interactive objects, which will be used in the virtual world.

Implementing the virtual world.
In this chapter, the virtual world is created and configured in the NeosVR application and visually programmed using a visual programming language called LogiX.

Chapter 2

Vector spaces

Vector spaces can be naively understood as a collection of objects with a clearly defined “nice” properties. These elements are called vectors and can be intuitively represented as arrows in a two or three-dimensional coordinate space. To precisely define vector spaces, a different concept from linear algebra needs to be explained first. This concept is called a field, which plays a fundamental part in the definition of vector space.

2.1 Fields

To avoid the restriction of using only a specific kind of numbers, like $\mathbb{Q}$ (rational numbers), $\mathbb{R}$ (real numbers) or $\mathbb{C}$ (complex numbers), the scalars are presented. Scalars can be thought of as any type of numbers ($\mathbb{Q}$, $\mathbb{R}$, $\mathbb{C}$), for which the following axioms hold.

(A) “To every pair, α and β, of scalars there corresponds a scalar $\alpha + \beta$, called the *sum* of α and β, in such a way that

  (1) addition is commutative, $\alpha + \beta = \beta + \alpha$,

  (2) addition is associative, $\alpha + (\beta + \gamma) = (\alpha + \beta) + \gamma$,

  (3) there exists a unique scalar 0 (called *zero*) such that $\alpha + 0 = \alpha$ for every scalar α, and

  (4) to every scalar α there corresponds a unique scalar $-\alpha$ such that $\alpha + (-\alpha) = 0$.

(B) To every pair, α and β, of scalars there corresponds a scalar $\alpha\beta$, called the *product* of α and β, in such a way that

  (1) multiplication is commutative, $\alpha\beta = \beta\alpha$,

  (2) multiplication is associative, $\alpha(\beta\gamma) = (\alpha\beta)\gamma$,

   (3) there exist a unique non-zero scalar 1 (called *one*) such that $\alpha 1 = \alpha$ for every scalar α, and
   (4) to every non-zero scalar α there corresponds a unique scalar α^{-1} (or $\frac{1}{\alpha}$) such that $\alpha\alpha^{-1} = 1$.

(C) Multiplication is distributive with respect to addition, $\alpha(\beta+\gamma) = \alpha\beta + \alpha\gamma$." [1]

In the axioms above, the existence of two binary scalar operations is assumed. Addition (symbolized as $+$) and multiplication (symbolized as $\cdot$). If these two operations are defined for a given set of numbers (scalars) $\mathcal{M}$ and the conditions above hold, the ordered triple $\mathcal{F} = (\mathcal{M}, +, \cdot)$ is called a field. [2]

2.2 Vector spaces

The vector spaces definition can now be introduced using the previously defined field. Using the scalars of a given field $\mathcal{F}$, the vector space is defined as follows.

Definition 2.2.1. *A vector space is a set $\mathcal{V}$ of elements called vectors satisfying the following axioms.*

(A) "To every pair, x and y, of vectors in $\mathcal{V}$ there corresponds a vector $x+y$, called the *sum*, in such a way that
   (1) addition is commutative, $x + y = y + x$,
   (2) addition is associative, $x + (y + z) = (x + y) + z$,
   (3) there exists in $\mathcal{V}$ a unique vector θ (called the *origin*) such that $x + \theta = x$ for every vector x, and
   (4) to every vector x in $\mathcal{V}$ there corresponds a unique vector $-x$ such that $x + (-x) = \theta$.

(B) To every pair, α and x, where α is a scalar and x is a vector in $\mathcal{V}$, there corresponds a vector αx in $\mathcal{V}$, called the *product* of α and x, in such a way that
   (1) multiplication by scalars is associative, $\alpha(\beta x) = (\alpha\beta)x$, and
   (2) $1x = x$ for every vector x.

(C) (1) Multiplication by scalars is distributive with respect to vector addition, $\alpha(x + y) = \alpha x + \alpha y$, and

(2) multiplication by vectors is distributive with respect to scalar addition, $(\alpha + \beta)x = \alpha x + \beta x$." [1]

Because of the definition, an example of a vector space can finally be presented. Let's start simple, and take a set of all real numbers $\mathcal{R}$. If the operations from the vector space definition are ordinary addition and multiplication, $\mathcal{R}$ becomes a vector space. This vector space will be called a one-dimensional real vector space and be written as $\mathcal{R}^1$ (one-dimensional because the elements of the vector space are individual numbers, and real because it is using real numbers). [1]

More interesting examples are $\mathcal{R}^2$ and $\mathcal{R}^3$, which represent sets of ordered doubles and triples, also called 2-tuple and 3-tuple (where n-tuple is an ordered set of n elements) of real numbers. Because the elements are not individual numbers but tuples, the ordinary addition and multiplication will not work. Instead, slightly modified versions of addition and multiplication are required. Let $x = (x_1, ..., x_n)$ and $y = (y_1, ..., y_n)$ be two random elements from $\mathcal{R}^2$ or $\mathcal{R}^3$ (so $n = 2$ or $n = 3$) and α be a scalar from field $\mathcal{R}$, the operations are defined as

$$x + y = (x_1 + y_1, ..., x_n + y_n), \tag{2.1}$$

$$\alpha x = (\alpha x_1, ..., \alpha x_n). \tag{2.2}$$

The vector space definition 2.2.1 also expects the existence of elements θ and $-x$, which are not yet defined for the 2 or 3-tuples. Luckily this is easily fixed, by the following intuitive definitions

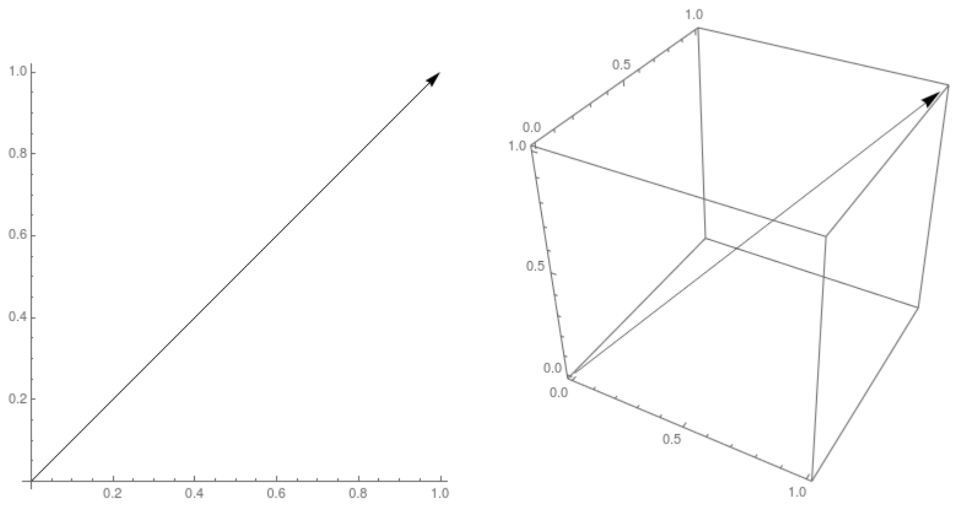
$$\theta = (0, ..., 0), \tag{2.3}$$

$$-x = (-x_1, ..., -x_n). \tag{2.4}$$

Thanks to the preparation above, the definition of vector space is fulfilled, therefore $\mathcal{R}^2$, and $\mathcal{R}^3$ are vector spaces. Both $\mathcal{R}^2$ and $\mathcal{R}^3$ are easily represented (see figure 2.1), and because of that will be used as examples throughout this thesis. [1]

The next logical step would be to show, that even higher dimension of $\mathcal{R}$ ($\mathcal{R}^4$, $\mathcal{R}^5$, $\mathcal{R}^6$, ...) are vector spaces. What may also be useful is to work with not only real but also rational and complex numbers. The good news is that there is not a need to prove each combination one by one. Both $\mathcal{Q}$, and $\mathcal{C}$ satisfy the definition of *field*, and the presented tuple addition and multiplication work in any dimension. This means that $\mathcal{R}^4$, $\mathcal{Q}^5$, $\mathcal{C}^6$, and even general $\mathcal{F}^n$ over a field $\mathcal{F}$ are all vector spaces.

New vector spaces over more complex elements (like matrices or polynomials) are also easy to introduce. The only requirement is to choose suitable field $\mathcal{F}$, and vector addition and scalar multiplication according to the definitions, this way the resulting set $\mathcal{V}$ over a field $\mathcal{F}$ is always a vector space.



(a) (b)

Figure 2.1: Representation of vector in $\mathcal{R}^2$ (a) and $\mathcal{R}^3$ (b)

2.3 Properties

Since the definition of vector spaces is written as short as possible, it says only the bare minimum. To work more efficiently with vector spaces and clarify things; that may not have been so obvious from the definition, the following theorems are offered. These theorems are direct implications and are based solely on the axioms from the vector space definition 2.2.1.

Theorem 2.3.1 (Zero Vector is Unique)**.** *"Suppose that $\mathcal{V}$ is a vector space. The zero vector θ, is unique."* [3]

Theorem 2.3.2 (Additive Inverses are Unique)**.** *"Suppose that $\mathcal{V}$ is a vector space. For each $x \in \mathcal{V}$, the additive inverse, $-x$, is unique."* [3]

Theorem 2.3.3 (Zero Scalar in Scalar Multiplication)**.** *"Suppose that $\mathcal{V}$ is a vector space and $x \in \mathcal{V}$. Then $0x = \theta$."* [3]

Theorem 2.3.4 (Zero Vector in Scalar Multiplication)**.** *"Suppose that $\mathcal{V}$ is a vector space and $\alpha \in \mathcal{F}$, where $\mathcal{F}$ is a field. Then $\alpha\theta = \theta$."* [3]

Theorem 2.3.5 (Additive Inverses from Scalar Multiplication)**.** *"Suppose that $\mathcal{V}$ is a vector space and $x \in \mathcal{V}$. Then $-x = -1x$."* [3]

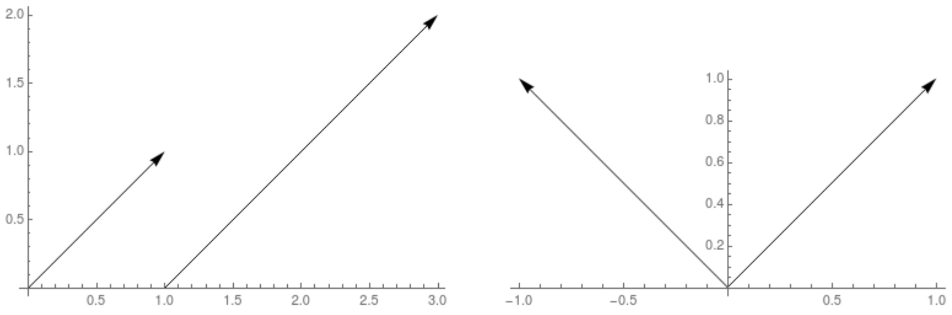
Theorem 2.3.6 (Scalar Multiplication Equals the Zero Vector)**.** *"Suppose that $\mathcal{V}$ is a vector space and $\alpha \in \mathcal{F}$, where $\mathcal{F}$ is a field. If $\alpha x = \theta$, then either $\alpha = 0$ or $x = \theta$."* [3]

Since the theorems assume only the existence of a vector field, all of them will apply to any vector space. Because of that, they will be now used freely in the rest of this thesis as if they were a part of the vector space definition (for the proofs of the individual theorems see [3]).

2.4 Vector algebra

In the axioms of vector spaces, a few simple vector operations were introduced. However, this is not everything that can be done with vectors. Here, the most fundamental vector operations are presented, which will be the ones implemented in the practical part.

Before that, a closer look at vectors is necessary. The current knowledge about vectors, based on the previous text, is that vector is something, which is an element of a vector space. For simplification, the vector space $\mathcal{R}^2$ is chosen as an example. What this means is that vectors from this vector space will be 2-tuples and be written in the following way $x = (x_1, x_2)$. As said previously, vectors from a 2-dimensional vector space can easily be visualized as a directed line segment (see figure 2.1 (a)). This line segment has a specific *magnitude*, which is the length of the vector, and a *direction*, which denotes in what direction is the vector pointing. For the visual representation of vector properties, see figure 2.2. [2, 4]



(a) Same direction, different magnitude (b) Same magnitude, different direction

Figure 2.2: Visual representation of vector properties

2.4.1 Vector equality

The first thing that can be done with vectors is to decide whether they are equal. For two vectors to be considered equal, they need to have the same direction and magnitude. What, however, is not required is that both vectors start at the same point. An example of this can be seen in figure 2.3, where both vectors start and end somewhere else, but are still considered equal. Another thing which may be seen on the figure 2.3 is that they are parallel. This is not an accident, and it is a common characteristic of all equal vectors.

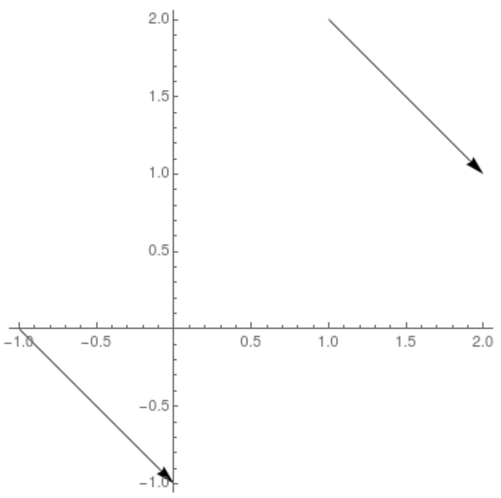


Figure 2.3: Equal vectors

Equal vectors are symbolized the following way $x = y$. Similarly, vectors that are not equal are written as $x \neq y$. [4]

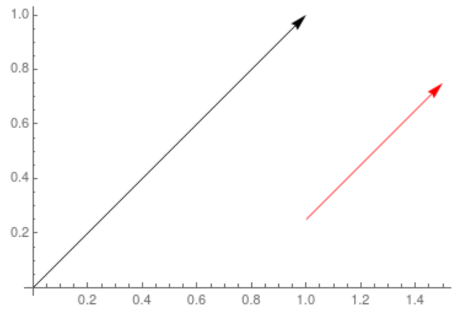
2.4.2 Scalar multiplication

The next operation that can be performed on vectors is called scalar multiplication. The good news is that this was a part of the vector space definition, so it is a known operation. Scalar multiplication takes a vector and scalar and increases or decreases the vector's magnitude by multiplying it by α. This operation can also be referred to as "scaling", which is the reason scalars are called the way they are. [4]

One thing to have in mind is the multiplication by a negative scalar. That is because it not only scales the vector but also changes the vector's direction by 180°. This can be easily explained using theorem 2.3.5. This theorem says that given a vector x, the following equation holds $-x = -1x$. Thanks to this and the associative property of scalar multiplication the multiplication by negative scalar can be written the following way:

$$-\alpha x = -1(\alpha x) \qquad \text{(where } \alpha \text{ is scalar)}$$
$$-1y = -y \qquad \text{(where } y \text{ is } \alpha x\text{)}$$

What this shows is that multiplication by $-\alpha$ can be done as multiplying the vector by α (which scales the vector) followed by making the additive inverse of the result (which changes the direction of the vector). Examples of both positive and negative scalar multiplication can be seen in figure 2.4.


(a) Vector x (black) and vector $0.5x$ (red)

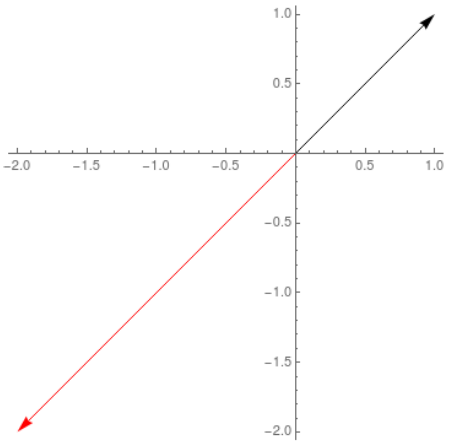

(b) Vector x (black) and vector $-2x$ (red)

Figure 2.4: Visualization of scalar multiplication

2.4.3 Vector addition and subtraction

Another known operation introduced as a part of the vector space definition is vector addition. This operation takes two vectors and creates a new one by summing the respective vector's components. The result of this operation also holds a geometrical significance. If given two vectors x and y, where vector y is shifted in a way that it starts where vector x ends, the resulting vector $x + y$ starts at the starting point of x and ends at the ending point of y thus forming a triangle. This is shown in figure 2.5. [4]

Vector subtraction is similar to this and can be explained by the following question. "What vector z must be added to y to give x? The vector z is defined to be the vector $x - y$." From this, the geometrical significance of the vector subtraction is obvious. A perhaps simpler way to introduce vector subtraction is to rewrite it as a vector addition. To achieve this theorem 2.3.5 is used again. This allows rewriting $x - y$ as $x + (-y)$. Because this option uses only already defined operations (vector addition and the additive inverse of a vector), it provides a simpler way to look at vector subtraction. [4]

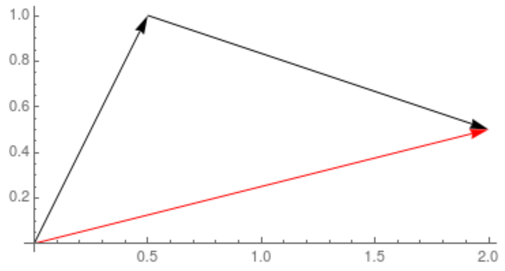


Figure 2.5: Representation of x (black), y (black) and $x + y$ (red)

2.4.4 Dot product

What is interesting about the vector space definition is that it does not say anything about multiplying vectors together. This may be because vector multiplication is slightly more complicated, and there are multiple ways to achieve it. The first approach to vector multiplication is an operation called the dot product, sometimes referred to as a scalar product. This operation takes two vectors and returns a scalar (hence scalar product) and is defined the following way,

$$x \cdot y = |x||y| \cos\theta, \tag{2.5}$$

where "θ is the angle between the two vectors when drawn from a common origin" and $|x|$ is the magnitude of vector x. [4]

One of the reasons why dot product is important is because it can tell if two vectors are perpendicular. Perpendicular vectors are an important concept of vector spaces, which will be explained later in the text. For two non-zero vectors x and y to be perpendicular, the following equation needs to be true,

$$x \cdot y = 0, \tag{2.6}$$

this is because $\cos\frac{\pi}{2} = 0$. [4]

There is also a second way to define the dot product. Given two vectors $x = (x_1, ..., x_n)$ and $y = (y_1, ..., y_n)$ the scalar product $x \cdot y$ can be written as:

$$x \cdot y = \sum_{i=1}^{n} x_i y_i \tag{2.7}$$

Both of the definitions above give the same results and are therefore equivalent. The benefit of the eq. (2.7) is that since it works directly with the vector's components which is mostly how the vectors are defined in algebraic contexts. Another advantage of having two equivalent definitions of the dot product is the ability to calculate otherwise unknown variables. A useful example of this is to find the angle θ between x and y. This can be achieved the following way:

$$\begin{aligned}
x \cdot y &= \alpha && \text{(calculate } \alpha \text{ using eq. (2.7))} \\
\alpha &= |x||y| \cos\theta && \text{(rewrite eq. (2.5) equal to } \alpha\text{)} \\
\cos\theta &= \frac{\alpha}{|x||y|} && \text{(express } \cos\theta\text{)} \\
\theta &= \arccos\frac{\alpha}{|x||y|} && \text{(express } \theta\text{)}
\end{aligned}$$

This introduces a way to easily calculate θ based only on the knowledge of the vector's components.

2.4.5 Cross product

Another way how vectors can be multiplied is by an operation called the cross product. Cross product is also sometimes referred to as a vector product. Similarly to dot product, the cross product takes two vectors, but instead of a scalar, it returns a vector (hence vector product). Cross product of vectors x and y is defined the following way,

$$x \times y = |x||y| \sin\theta \; E, \tag{2.8}$$

where E is a unit vector (unit vector E is any vector for which $|E| = 1$ is true) perpendicular to x and y. The above definition implies a few important things. First, if the directions of vectors x and y are the same, or opposite ($\theta = 0$ or $\theta = \pi$), the resulting vector is zero (this is because $\sin 0 = 0$ and $\sin\pi = 0$). Second, for the unit vector E to be perpendicular to both x and y, the dimension of the vector space needs to be at least 3. The reason for this is because it is impossible to find a perpendicular vector E to both x and y in vector spaces of dimension two and lower (apart from the stated special case where both x and y are parallel). For this reason, all of the vectors used to explain the cross product will be a part of a three-dimensional vector space. [4]

Cross product is an operation used in physics, computer programming, mechanics, and more. Shown here will however be its geometrical importance, which can be described by the following statement. "Given any two nonparallel vectors x and y, a third vector z may be constructed as follows: When translated so that they have a common origin, the two vectors x, y form two sides of a parallelogram. Vector z is defined to be perpendicular to the plane of this parallelogram with magnitude equal to the area of the parallelogram." An example of this can be seen in figure 2.6. [4]

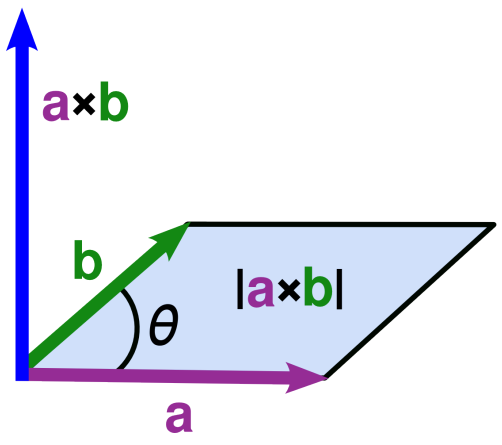


Figure 2.6: Geometrical significance of cross product [5]

2.5 Matrix transformations

In section 2.4, several vector operations were introduced, which allowed to add and subtract vectors, multiply vectors, and more. All of these operations are crucial, but all of them create a new object (be it a vector or a scalar). The only one which allows changing the actual vector (transforms it) is scalar multiplication, which has strict limitations. The main limitation is that it only allows changing the magnitude of the vector (or in case of negative scalar multiplication also the direction but in a precisely defined way). So, for example, rotating the vector by 90° is impossible with the currently available operations. To achieve this, something more complex is necessary. The way presented here is called matrix transformations, which use matrices to transform vectors (both their magnitude and direction).

Definition 2.5.1. *"Let m, n* $\in \mathbb{N}$ *and* $\alpha_{i,j} \in \mathcal{F}$ *for all* $i \in \{1, ..., m\}$ *and* $j \in \{1, ..., n\}$. *The rectangle shaped array*

$$\mathbb{A}_{m,n} = \begin{pmatrix} \alpha_{1,1} & \alpha_{1,2} & \cdots & \alpha_{1,n} \\ \alpha_{2,1} & \alpha_{2,2} & \cdots & \alpha_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ \alpha_{m,1} & \alpha_{m,2} & \cdots & \alpha_{m,n} \end{pmatrix}$$

of m rows and n columns is called a matrix of type $m \times n$ *with entries from the field* $\mathcal{F}$*."* [6]

For this to be useful, an operation using matrices needs to be presented, this operation is called *matrix multiplication* and is defined the following way.

Definition 2.5.2. *"Let* $m, n, p \in \mathbb{N}, \mathbb{A} \in \mathcal{F}^{m,n}$ *matrix with entries* $\alpha_{i,j}$ *and* $\mathbb{B} \in \mathcal{F}^{n,p}$ *matrix with entries* $\beta_{i,j}$. *The product of matrices* $\mathbb{A}$ *and* $\mathbb{B}$ *is matrix* $\mathbb{D} \in \mathcal{F}^{m,p}$ *with entries* $\delta_{i,j}$ *obtained as*

$$\delta_{i,j} = \sum_{k=1}^{n} \alpha_{i,k}\beta_{k,j},$$

and written as $\mathbb{A}\mathbb{B} = \mathbb{D}$*."* [2]

Before continuing with the matrix transformations, three important definitions related to matrices are presented, which will prove useful in the later sections.

Definition 2.5.3. *"Let* $n \in \mathbb{N}$. *Every bijection* $\pi : \{1, 2, ..., n\} \to \{1, 2, ..., n\}$ *is called permutation of the set* $\{1, 2, ..., n\}$. *The set of all permutations of* $\{1, 2, ..., n\}$ *is denoted by* S_n*."* [7]

Definition 2.5.4. *“Let $\pi \in S_n$. Every pair $(\pi(i), \pi(j))$ such that*

$$i < j \wedge \pi(i) > \pi(j), \quad i, j \in \{1, ..., n\}$$

is called inversion in π. The number sgn π defined as $(-1)^{I_\pi}$, where I_π is the number of all inversions, is called the signature of permutation π.” [7]

Definition 2.5.5. *“Let $\mathbb{A} \in \mathcal{F}^{n,n}$ be a matrix with entries $\alpha_{i,j}$. The determinant of matrix $\mathbb{A}$ is a number from $\mathcal{F}$ defined as*

$$\det \mathbb{A} = \sum_{\pi \in S_n} sgn\ \pi \cdot \alpha_{1,\pi(1)} \cdot \alpha_{2,\pi(2)} \cdot \dots \cdot \alpha_{n,\pi(n)}.\text{” [7]}$$

Now that these definitions are out of the way, it is possible to return to the matrix multiplication, which provides a way to multiply matrices together but says nothing about vectors. Luckily this can be easily fixed by writing the vectors as column matrices. Given a vector $x = (x_1, ..., x_n)$ it is possible to rewrite it as $n \times 1$ matrix $\mathbb{X}$ like this.

$$\mathbb{X} = \begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix} \tag{2.9}$$

With this knowledge, it is now possible to multiply vectors with matrices. The vectors from vector space $\mathcal{R}^2$ are chosen to serve as examples for matrix transformations shown here, but the transformation matrices can be easily adapted to work in any dimension.

First, it is important to choose the transformation matrix of the correct type so that the operation $\mathbb{A}\mathbb{X}$ (where $\mathbb{A}$ is the transformation matrix and $\mathbb{X}$ is the vector written as a column vector) is well defined. Since the vectors from vector space $\mathcal{R}^2$ have two dimensions, and it is expected that the resulting vector will also have two dimensions, the resulting transformation matrix will be 2×2. With this, all of the required theory is defined, and a few interesting matrix transformations are presented.

2.5.1 Scaling

To scale vectors, matrix

$$\mathbb{A} = \begin{pmatrix} \alpha & 0 \\ 0 & \beta \end{pmatrix}$$

is used, which scales the vector by α in the direction of axis x and by β in the direction of axis y. In case both α and β are equal to 1, an identity is obtained (identity is a special kind of transformation that does not change the vector). [2]

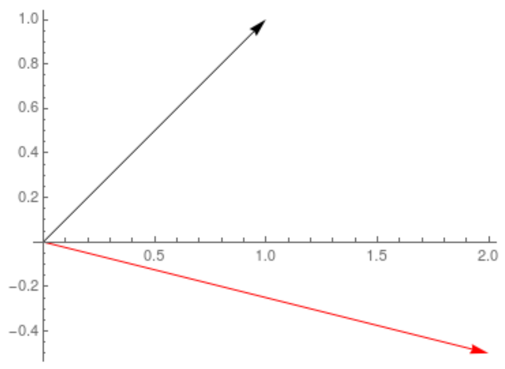


Figure 2.7: Scaling transformation where $\alpha = 2$ and $\beta = -0.5$

2.5.2 Shearing

To shear vectors, matrix

$$\mathbb{A} = \begin{pmatrix} 1 & \lambda \\ 0 & 1 \end{pmatrix}$$

is used, “which maintains the vectors in the direction of the y-axis and in the direction of the x-axis increases them by λ multiple of the second component” (shears them parallel to the x-axis). To shear vectors by λ parallel to the y-axis a similarly looking matrix

$$\mathbb{A} = \begin{pmatrix} 1 & 0 \\ \lambda & 1 \end{pmatrix}$$

can be used. [2]

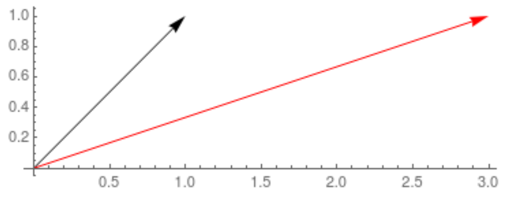


Figure 2.8: Shearing transformation parallel to the x-axis where $\lambda = 2$

2.5.3 Rotation

To rotate the vectors around the origin θ anti-clockwise, matrix

$$\mathbb{A} = \begin{pmatrix} \cos\varphi & -\sin\varphi \\ \sin\varphi & \cos\varphi \end{pmatrix}$$

is used, which rotates the vector by angle φ. [2]

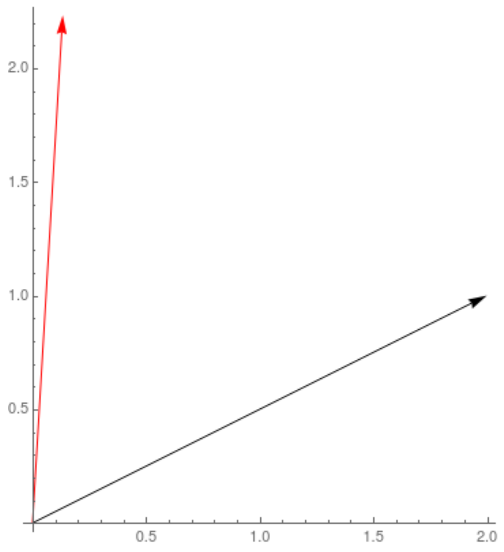


Figure 2.9: Rotation transformation where $\varphi = 60°$

2.6 Linear dependence

Another essential concept of vector spaces is linear dependence and linear independence. This property is described over a set of vectors and is defined the following way.

Definition 2.6.1. *"Let $(x_1, ..., x_n)$ be a set of vectors from vector space $\mathcal{V}$. The set $(x_1, ..., x_n)$ is called linearly dependent if there exists a corresponding set $(\alpha_1, ..., \alpha_n)$ of scalars, not all zero, such that*

$$\sum_{i=1}^{n} \alpha_i x_i = \theta.$$

If, on the other hand, $\sum_i \alpha_i x_i = \theta$ implies that $\alpha_i = 0$ for each i, the set $(x_1, ..., x_n)$ is linearly independent." [1]

What this means is that from a linearly independent set of vectors, it is impossible (using scalar multiplication and vector addition) to create a zero vector θ. In a linearly dependent set of vectors, on the other hand, has to exist at least one way to gradually add and scale vectors (this process is called *linear combination*) to obtain zero vector θ.

An example of a linearly dependent set of vectors from vector space $\mathcal{R}^2$ is (x, y, z), where $x = (1, 1), y = (0.5, -0.5)$ and $z = (2, 0.5)$. To show that this set is linearly dependent according to the definition, vector z is expressed as a linear combination of x and y.

$$\alpha_1(1, 1) + \alpha_2(0.5, -0.5) = (2, 0.5) \tag{2.10}$$

Rewritten as a system of equations:

$$\begin{aligned} 1\alpha_1 + 0.5\alpha_2 &= 2 \\ 1\alpha_1 - 0.5\alpha_2 &= 0.5 \end{aligned} \tag{2.11}$$

Expressed α_1 and solved for α_2:

$$\begin{aligned} \alpha_1 &= 2 - 0.5\alpha_2 \\ \alpha_2 &= -1 + 2\alpha_1 \\ &= -1 + 4 - \alpha_2 \\ &= -0.5 + 2 \\ &= 1.5 \end{aligned} \tag{2.12}$$

Solved for α_1:

$$\begin{aligned} \alpha_1 &= 2 - 0.5\alpha_2 \\ &= 2 - 0.75 \\ &= 1.25 \end{aligned} \tag{2.13}$$

With the expressed α_1 and α_2, z can finally be written as a linear combination of vectors x and y like so:

$$1.5(1, 1) + 1.25(0.5, -0.5) = (2, 0.5) \tag{2.14}$$

With this, only a last step remains to prove that set (x, y, z) is linearly dependent, and that is to construct a set of scalars $(\alpha_1, \alpha_2, \alpha_3)$ from which α_1 and α_2 are already known. Since the parameters α_1 and α_2 were chosen so that they give vector z, vector θ can be obtained (by definition) by subtracting vector z. The parameter α_3 will, therefore, be -1. Linear dependence can be then easily proven by writing the known variables in the definition of a linearly dependent set of vectors like so:

$$\begin{aligned} \sum_{i=1}^{3} \alpha_i x_i &= \alpha_1 x + \alpha_2 y + \alpha_3 z \\ &= 1.25(1, 1) + 1.5(0.5, -0.5) + (-1)(2, 0.5) \\ &= (1.25, 1.25) + (0.75, -0.75) + (-2, -0.5) \\ &= (0, 0) = \theta \end{aligned} \tag{2.15}$$

Because the set of not all zero scalars $(\alpha_1, \alpha_2, \alpha_3)$ exists, the vector set (x, y, z) is linearly dependent. A visualization of this example can be seen in figure 2.10.

Another way to obtain the information whether a given set is linearly dependent or independent is to use the determinant of a matrix consisting of column vectors. Because the determinant is defined only for the square matrices, this method can be used only when the size of the set is the same as the dimension of the vector space. If this condition holds, the following theorem can be used. [2]

Theorem 2.6.1. *Let $\mathbb{A} \in \mathcal{F}^{n,n}$. The set of columns of matrix $\mathbb{A}$ is linearly independent if and only if det $\mathbb{A} \neq 0$.* [2]

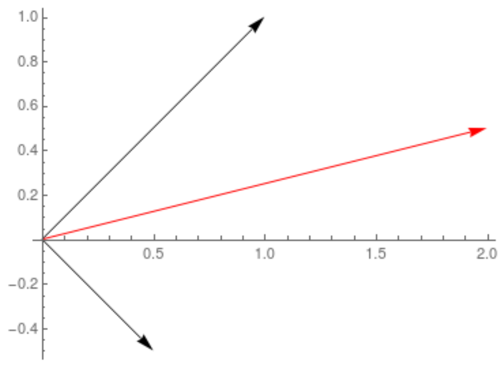


Figure 2.10: Set of linearly dependent vectors x (black), y (black) and z (red)

2.7 Bases

In the previous section about linear dependence, the main focus was to determine whether a given set of vectors was linearly dependent or linearly independent. An intuitive observation from the definition of linear dependence was that by adding new vectors into an already existing set of vectors, which is linearly independent, there has to come a moment after which the set becomes linearly dependent. But what if the goal was to get a linearly independent set from which every other vector from the vector space could be constructed by linear combination? The concept that was just intuitively described is called a *basis* and can be formally defined the following way.

Definition 2.7.1. *"A basis in a vector space $\mathcal{V}$ is a set $\mathcal{X}$ of linearly independent vectors such that every vector in $\mathcal{V}$ is a linear combination of elements of $\mathcal{X}$."* [1]

The concept of basis is not anything new and is naturally used when working with two or three-dimensional coordinate systems, where the basis vectors simply copy the axes. Let's use the three-dimensional coordinate system (which, according to the definition, is a fully valid vector space) as an example. The simplest basis for such a vector space is $((1,0,0),(0,1,0),(0,0,1))$ since every other vector can be described as $\alpha_1(1,0,0)+\alpha_2(0,1,0)+\alpha_3(0,0,1)$. This type of basis (composed of only unit vectors) is called a *standard basis.* [2]

Because the basis allows expressing every possible vector as a linear combination of basis vectors, it is possible to write vectors with relation to the given basis $\mathcal{X}$ like so.

Definition 2.7.2. *"Let $\mathcal{X} = (x_1, ..., x_n)$ be a basis of vector space $\mathcal{V}^n$ and vector z from $\mathcal{V}^n$ given as a linear combination*

$$z = \sum_{i=1}^{n} \alpha_i x_i.$$

- *Scalar α_i is called the i-th coordinate of vector z with respect to basis $\mathcal{X}$.*
- *The coordinates of z with respect to basis $\mathcal{X}$ are understood as the n-tuple*

$$(z)_{\mathcal{X}} = (\alpha_1, ..., \alpha_n)."\ [8]$$

To obtain the coordinates for a given vector $z = (z_1, ..., z_n)$, a system of equations needs to be solved (similarly to eq. (2.10)). A special case happens when working with a standard basis $\mathcal{E}$, where the vector's components correspond to the vector's coordinates with respect to the standard basis $\mathcal{E}$.

$$z = (z)_{\mathcal{E}} = (z_1, ..., z_n) \tag{2.16}$$

With the definition of the basis, it is now possible to define one term, which was so far understood only intuitively. This term is *dimension* and can be formally defined the following way:

Definition 2.7.3. *"The dimension of a finite-dimensional vector space $\mathcal{V}$ is the number of elements in a basis of $\mathcal{V}$."* [1]

2.7.1 Cramer's rule

The method for obtaining coordinates show here is completely valid but is nearly impossible to algorithmize. For this reason, a different way is commonly used on computers when solving systems of equations called Gaussian elimination. This method performs well but is still not as simple to implement, which is why a different approach to solving systems of equations is chosen called the Cramer's rule.

Cramer's rule works with systems of n equations and n unknowns and solves a problem in the form of $\mathbb{A}\mathbb{X} = \mathbb{B}$, where $\mathbb{A}$ is a $n \times n$ matrix with non-zero determinant and $\mathbb{X}$ is the column vector describing the solution for the system. On condition that the system has only one solution, this solution can be obtained as

$$\mathbb{X}_i = \frac{\det \mathbb{A}_i}{\det \mathbb{A}}, \quad i = \{1, ..., n\} \tag{2.17}$$

where $\mathbb{A}_i$ is a "matrix obtained from $\mathbb{A}$ by replacing the i-th column of $\mathbb{A}$ by the column vector $\mathbb{B}$". [9]

This provides a way to solve a system of equations, which can be easily algorithmized and can be used to determine the coordinates of a given vector x relative to the basis $\mathcal{X}$.

2.8 Orthogonality

When talking about vectors in section 2.4, two vector properties were introduced: magnitude and direction. What, however, was not said is how these values can be obtained from vectors defined as $x = (x_1, ..., x_n)$.

The general formula that measures the distance between two points $a = (a_1, x_2)$ and $b = (b_1, b_2)$ in a coordinate space $\mathcal{R}^2$ is $\sqrt{(a_1 - b_1)^2 + (a_2 - b_2)^2}$. The same slightly modified formula can be used to determine the magnitude of the vector. Having vector $x = (x_1, ..., x_n)$ from a n-dimensional vector space $\mathcal{R}^n$ the vector's magnitude is defined the following way:

$$|x| = \sqrt{x_1^2 + ... + x_n^2} \tag{2.18}$$

This distance measures the magnitude of vector x from the origin θ. What this means is that it could have also been written as,

$$|x - \theta| = \sqrt{(x_1 - 0)^2 + ... + (x_n - 0)^2}, \tag{2.19}$$

from which the relationship with the definition of the geometrical distance between points is apparent. [1]

When talking about dot product in subsection 2.4.4, the way to calculate the angle between vectors x and y was shown. When working with vectors, however, it is often more helpful to work with the cosines of the angles than the angles themselves.

What was also said in subsection 2.4.4 was, that vectors for which $x \cdot y = 0$ is true are important. Apart from being perpendicular, they are also called *orthogonal.* "Orthogonality is the most important relationship among the vectors" because it ensures a lot of significant properties. [1]

Orthogonality can also be generalized for a whole set of vectors $(x_1, ..., x_n)$, where each vector is orthogonal to one another. This can be written as,

$$\forall i, j \in \{1, ..., n\}, i \neq j : x_i \cdot x_j = 0. \tag{2.20}$$

An example of an orthogonal set of three vectors is in figure 2.6, where the resulting vector $a \times b$ is created as an orthogonal vector to already orthogonal vectors a and b. An essential characteristic of every orthogonal set of non-zero vectors is that it is always linearly independent (but not the other way around).

A useful application of orthogonality is the Pythagorean theorem which is defined the following way:

$$|x + y|^2 = |x|^2 + |y|^2 \tag{2.21}$$

This provides a more simplistic way to obtain the magnitude of the sum of vectors. [2]

Chapter 3

Virtual reality

"Virtual reality is a computer-generated 3D environment that allows experiencing virtual worlds similarly as the real ones." These worlds can represent any existent or non-existent place such as the Grand Canyon, the Moon, or the square world of Minecraft. [10]

To be able to work with virtual reality a specifically designed glasses (often referred to as a headset), are required. These glasses commonly contain two displays (one for each eye) displaying content to the user trying to mimic human vision. Thanks to this, the user can visually experience similar events as if he was in the real world. Apart from the headset, virtual reality also uses specific controllers that track the movement of the user's hands. Because of this, the user can physically interact with the objects merely by hand movements, which increases the overall immersion of virtual reality. An example of the virtual reality headset with two controllers can be seen in the figure 3.1. [10]

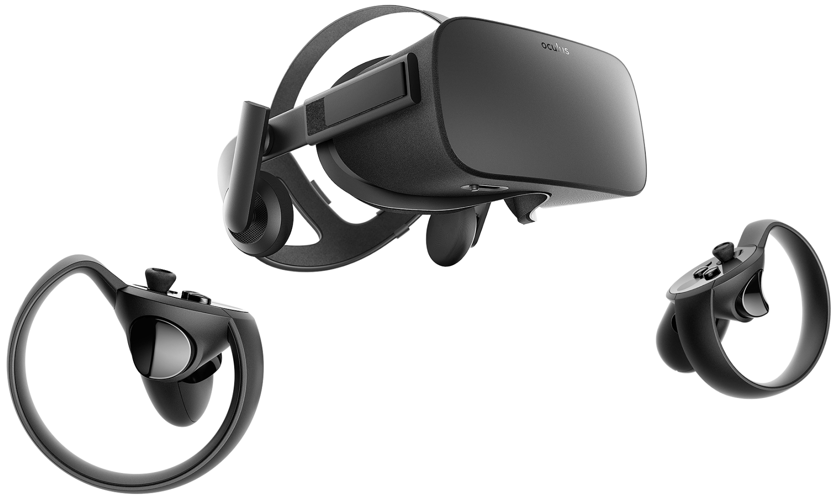


Figure 3.1: Virtual reality headset with two controllers (Oculus Rift) [11]

3.1 Uses of virtual reality

Virtual reality is currently used in many areas such as military, healthcare, and construction, but there are two which can be described as the most common.

First is the entertainment, which allows users to experience various enjoyable experiences like attending a virtual concert, watching a movie in a cinema-like environment, playing games, exercising body or soul, and more. [10]

Figure 3.2: Fishing game in virtual reality (Real VR Fishing) [12]

The second is education, which may have even higher potential because it allows students to easily visualize important concepts instead of relying on the classical 2D approaches (such as textbooks). It also allows teachers to host virtual lectures for a higher number of students and provides new ways of teacher–student interactions. [10, 13]

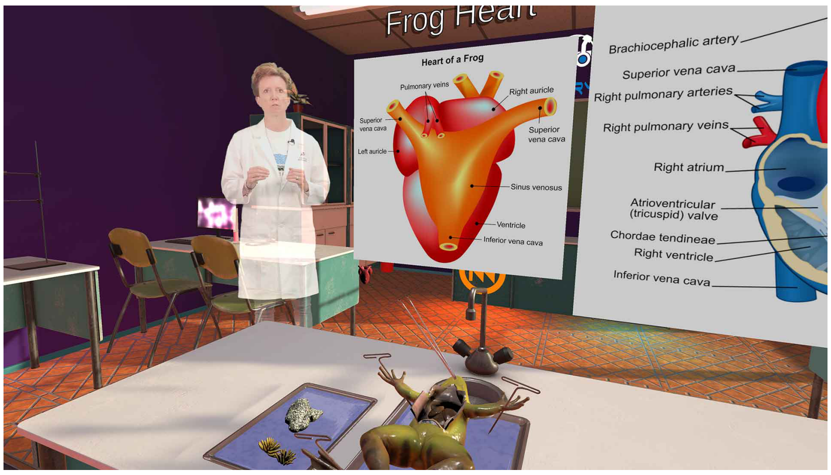


Figure 3.3: Dissection of a frog in virtual reality (VictoryXR) [14]

3.2 Developing for virtual reality

The development of worlds for virtual reality is similar to developing a game (this is true for both entertainment and non-entertainment worlds). The reason for this is that the world needs to know where the user is looking (getting user input). Next, it needs to show the appropriate scene (rendering). And all of this needs to happen multiple times per second to give the user a feeling of immersion (loop update). The three concepts just described here (and many more) are fundamental for game development, and are natively supported in every game engine, which is why game engines are commonly used when developing worlds for virtual reality.

Currently, there are two game engines the virtual worlds are commonly created in, Unity [15] and Unreal Engine [16]. Both these engines have their pros and cons but are capable of producing virtual worlds of top quality. A little downside of using a game engine is, that while it takes care of the low-level principles (getting the user input, updating/rendering the scene), the high-level principles are left to the developer (defining the physics and behavior of the objects). [10]

A slightly different approach to building virtual worlds provides an application called NeosVR [17]. This application itself runs in virtual reality and allows developers to create virtual worlds by simply placing predefined/custom objects into the scene, changing the object's properties, and overwriting the object's default behavior. Because of this, the developers can focus on building the virtual world instead of concerning themselves with the high-level principles the game engine would require them to do.

The approach that was chosen for the implementation of the practical part of this thesis is the latter one utilizing the NeosVR application. For this reason, the following sections will describe only the concepts used in this approach.

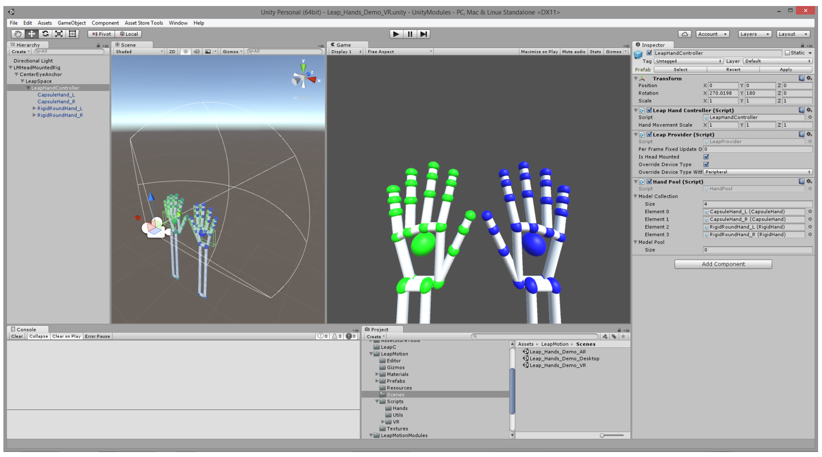


Figure 3.4: Developing for VR in Unity [18]

3.3 NeosVR

As revealed in the previous section, the NeosVR application provides a toolset specifically designed for the development of virtual worlds. Another thing that NeosVR offers is the possibility to explore worlds created by other users. Because of this, NeosVR is used by both developers, which create new virtual worlds, and consumers, which explore already existing virtual worlds and interact with them. Both of these activities can be done entirely in virtual reality or, in case the user does not possess VR headset or does not want to use it, in screen mode.

Downloading and installing NeosVR is simple because it is a part of Steam's [19] marketplace. All that is required is to create a Steam account, claim the application, and the rest like downloading, installing, and configuring is taken care of by the Steam's client.

Once NeosVR is installed and running, the user appears in a simple virtual world and can decide on what to do next. The user is also provided with an interactive menu (referred to as a *belt menu*), which provides actions such as exiting the game, creating a new world, opening the inventory, and more. The belt menu can be seen in the figure 3.5.

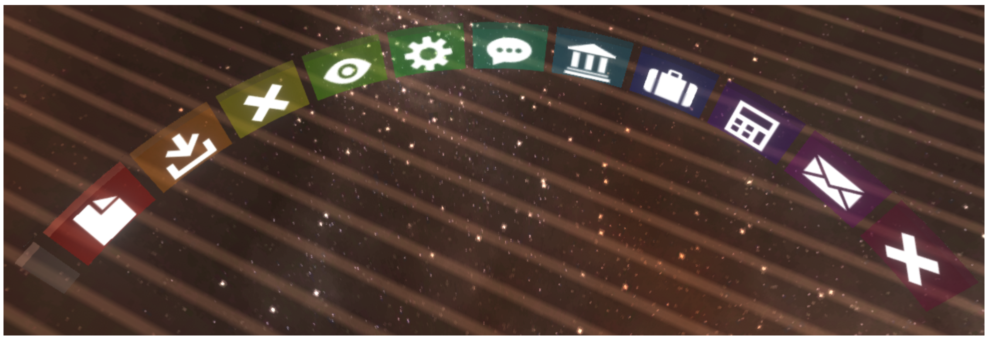


Figure 3.5: Belt menu inside of NeosVR

The first thing that the user can do in NeosVR is to join an already existing world. There are multiple ways to achieve this. By clicking on one of the world orbs above the "Currently active sessions" platform, which is present in the starting virtual world. By browsing the world catalog (obtainable by clicking the *building* icon in the belt menu and selecting "Browse Worlds"). Or by vising the content hub (accessible by clicking the *building* icon in the belt menu and selecting "Content Hub"), which offers worlds categorized into individual sections (this can be seen in the figure 3.6).

The second thing that the user can do is to start building new virtual worlds. Because this aspect of NeosVR is more complicated and is vital for this thesis, it will be explained in greater detail in the following subsections.

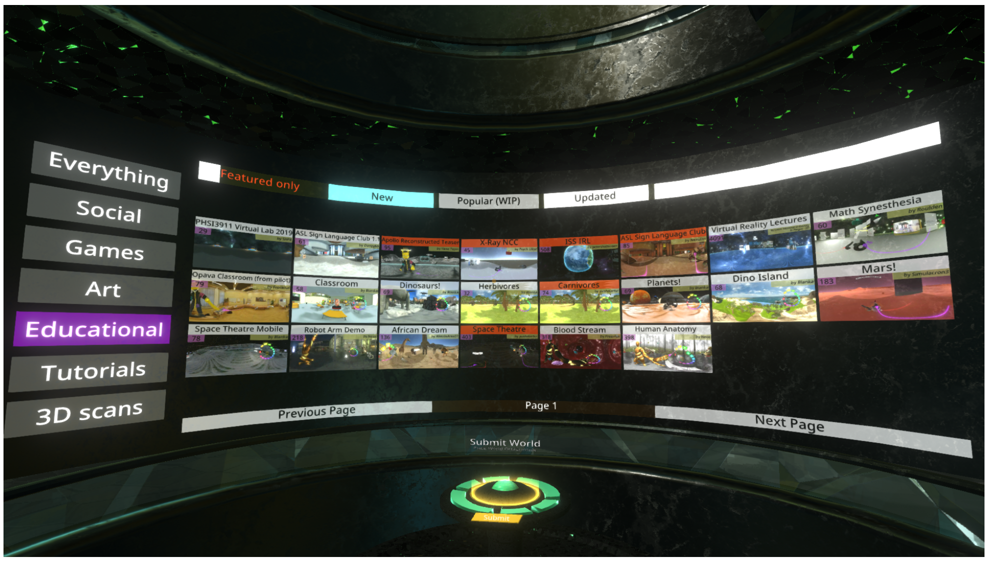
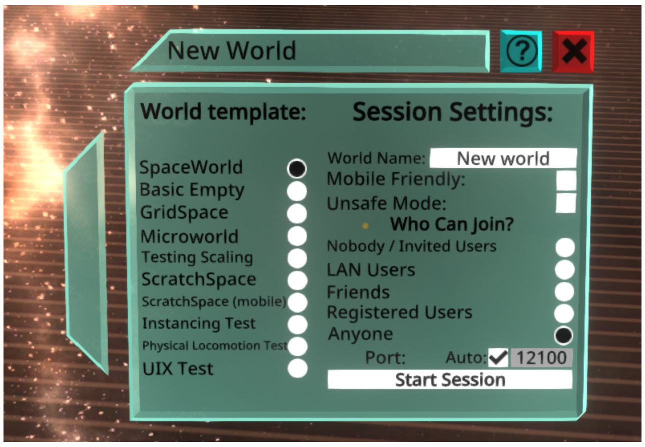


Figure 3.6: Content hub allowing to browse and enter worlds

3.3.1 Creating new worlds

The creation of worlds in NeosVR is simple and can be described similarly as creating a new document in a text editor. Firstly it is required to click the *file* icon in the belt menu, which brings up a configuration of the new world (this can be seen in the figure 3.7). Here the user can configure information like the name of the world, who can join the world and can also select from a few predefined world templates, which contain some already preset objects (such as the ground or the skybox). Once the user is happy with the selections, the "Start Session" button can be pressed, which, after a short loading, teleports the user into the newly created world.


New World

World template:
- SpaceWorld ●
- Basic Empty ○
- GridSpace ○
- Microworld ○
- Testing Scaling ○
- ScratchSpace ○
- ScratchSpace (mobile) ○
- Instancing Test ○
- Physical Locomotion Test ○
- UIX Test ○

Session Settings:
- World Name: New world
- Mobile Friendly: ☐
- Unsafe Mode: ☐

Who Can Join?
- Nobody / Invited Users ○
- LAN Users ○
- Friends ○
- Registered Users ○
- Anyone ●

Port: Auto: ✓ 12100

Start Session

Figure 3.7: New world configuration

3.3.2 Filling up the worlds

Creating new objects in NeosVR is also really intuitive. The user can either import new 3D objects from the computer (this is explained in greater detail in [10]) or use any of the already created objects in the NeosVR's inventory.

The inventory can be accessed by clicking the *suitcase* icon in the belt menu and is divided into two sections: public and private. The public section contains useful items like tools, materials, 3D models, and more, created either by the NeosVR's developers or by other users. The private section, on the other hand, provides a safe space for the user to store worlds and personally created objects. It also allows storing items from other worlds to make them quickly accessible. The inventory can be seen in the figure 3.8.

The desired item can be obtained by double-clicking its icon. This creates a copy of that item and places it in front of the user. This item can be then grabbed and dragged anywhere in the scene or used accordingly based on its type.

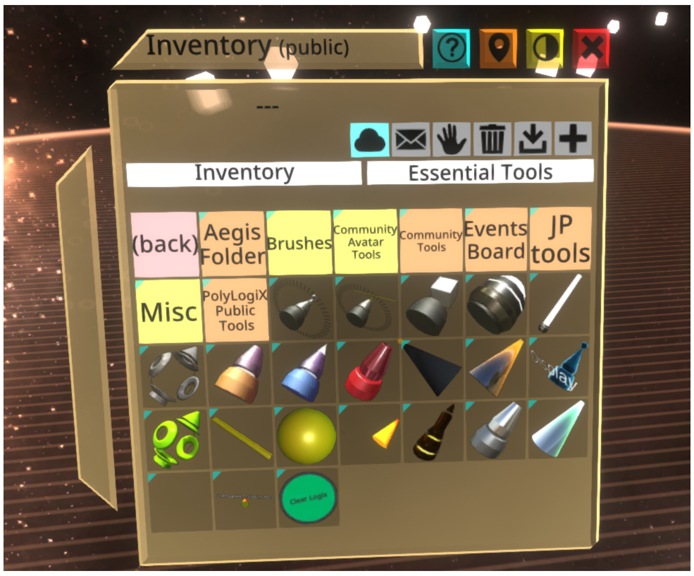
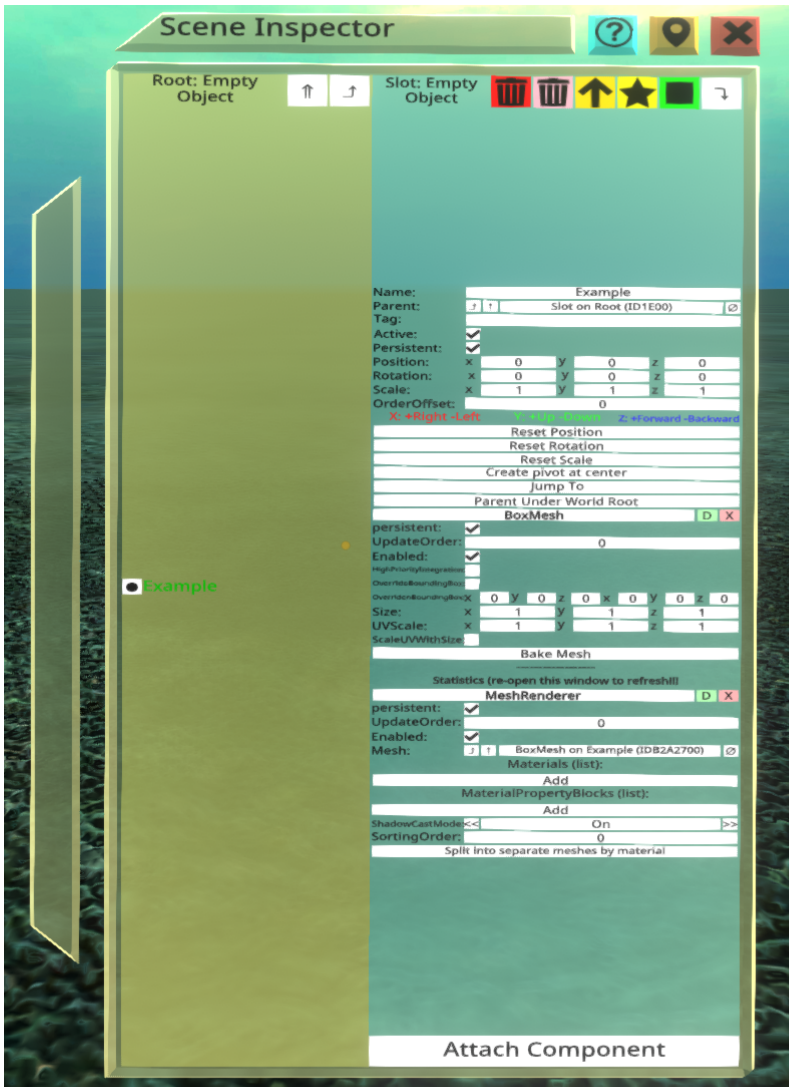


Figure 3.8: The inventory showcasing tools in the public section

3.3.3 The Inspector

The good thing about NeosVR is that it treats all of the objects inside the world equally. Because of this, every item inside of NeosVR (such as skybox, ground, tools) can be created from scratch. Objects in NeosVR are called *Slots* and are described by the following information: *name*, *parent*, *tag*, whether the slot is *active* and *persistent*, and by its *position*, *rotation*, and *scale* related to the parent.

To view these values, and more importantly, edit them, NeosVR uses something called the *inspector*. To open the inspector user needs to use the *Developer Tooltip*, which can be found in the public inventory inside the *Essential Tools* folder. Upon spawning the tooltip into the scene by double-clicking, the tool can be equipped by double-clicking the side button on the controller, which positions the controller into the user's hand. With the Developer Tooltip in hand, it is possible to open the inspector by clicking the controller menu button and selecting the "Open Inspector" option. The inspector can also be opened while in the screen mode by clicking the I key. With the inspector opened, the user is shown a dialog similar to the one in the figure 3.9. [10]



Name: Example
Parent: Slot on Root (ID1E00)
Tag:
Active: ✓
Persistent: ✓
Position: x 0 y 0 z 0
Rotation: x 0 y 0 z 0
Scale: x 1 y 1 z 1
OrderOffset: 0

BoxMesh D X
persistent: ✓
UpdateOrder: 0
Enabled: ✓
HighPriorityIntegration:
OverrideBoundingBox:
OverridenBoundingBox: x 0 y 0 z 0 x 0 y 0 z 0
Size: x 1 y 1 z 1
UVScale: x 1 y 1 z 1
ScaleUVWithSize:
Statistics (re-open this window to refresh!!)

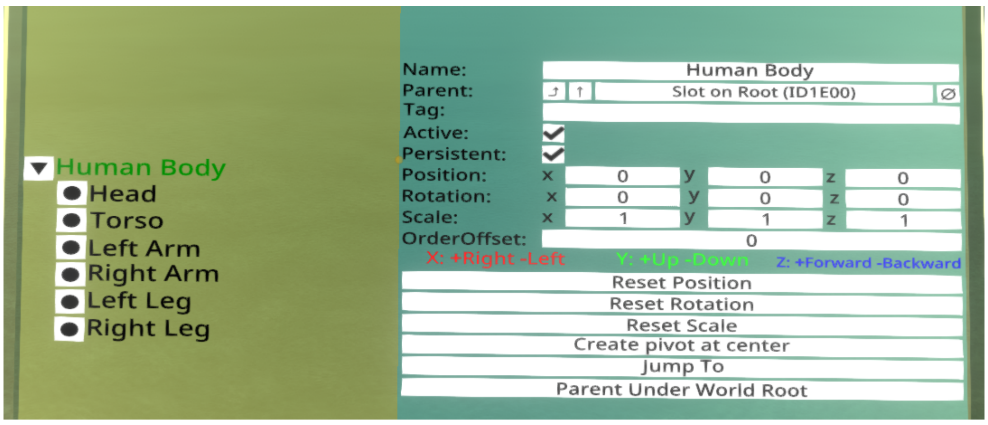
MeshRenderer D X
persistent: ✓
UpdateOrder: 0
Enabled: ✓
Mesh: BoxMesh on Example (IDB2A2700)
Materials (list):
MaterialPropertyBlocks (list):
ShadowCastMode: << On >>
SortingOrder: 0

Figure 3.9: The inspector for the Example slot with two components

The inspector is divided into two columns, each displaying a different set of information. The left side shows the hierarchy of the world (or of the object the inspector was opened for), whereas the right side shows the information related to the selected slot and can be used to edit the individual fields. It can also be used to destroy or duplicate the slot (by clicking the colored buttons in the top-right corner) and to attach, remove, or modify the slot's *components*.

Components are another concept that is essential for developing virtual worlds in NeosVR. As described before, slots contain only very general information (like name and position), but this is not enough to, for example, render the object on the screen. For this to happen, the slot would need to have a defined mesh (the object's skeleton), a material (the object's skin), and a renderer (which renders the result on the screen). The three examples described above are all components and would, upon attaching to a slot and correctly configuring, render the object. Except rendering, there are a lot of different areas the components can help with like, handling the user input (buttons, input fields), periodically changing the object (spinning, rotating), controlling the lighting and particle effects, and more.

The last thing the inspector can be used for is to change the hierarchy of the slots in the world. This is particularly helpful when working with an object that is composed of multiple slots because all changes applied to the parent slot (rotation, movement) are applied to children's slots as well. An example of this is a model of the human body, which is composed of several parts (head, torso, arms, legs) that move whenever the whole body moves. The slot's hierarchy can be changed by grabbing the selected slot on the left side of the inspector and dragging it under the new parent. An example of the Human Body hierarchy can be seen in the figure 3.10.



▼ Human Body
- Head
- Torso
- Left Arm
- Right Arm
- Left Leg
- Right Leg

Name: Human Body
Parent: Slot on Root (ID1E00)
Tag:
Active: ✓
Persistent: ✓
Position: x 0 y 0 z 0
Rotation: x 0 y 0 z 0
Scale: x 1 y 1 z 1
OrderOffset: 0
X: +Right -Left Y: +Up -Down Z: +Forward -Backward

Figure 3.10: The Human Body hierarchy

3.3.4 LogiX

LogiX is a visual programming language specially developed to be used in NeosVR. It allows users to define the behavior of objects, which would be hard or impossible to achieve by using the components. To use LogiX, the user needs to equip the *LogiX Tooltip* first. This tool is located at the same place as the Developer Tooltip, inside the Essential Tools folder in the public inventory. Once equipped, the user can open the nodes menu (similar to the one shown in the figure 3.11) by pressing the controller menu button and selecting the "Node Browser" option (or by pressing the C key while in the screen-mode). With the nodes menu opened, the user can choose any node by double-clicking it, which loads the LogiX Tooltip. Once loaded, the user can use the tool to create the selected node by double-clicking.

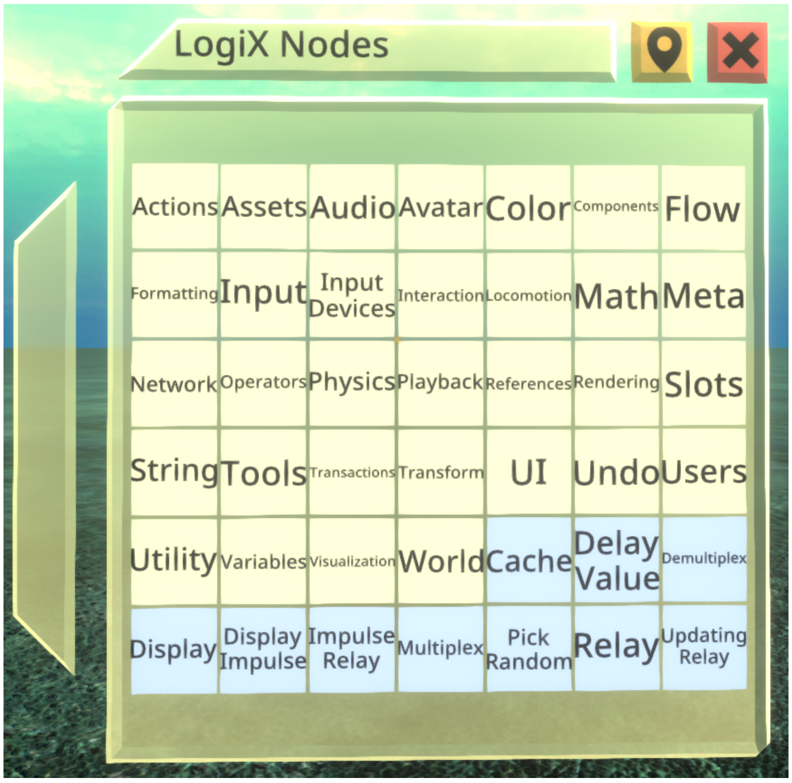


Figure 3.11: LogiX nodes menu

Nodes are the main building blocks of LogiX. Each LogiX node has a name (to describe what it does) and zero to many inputs and outputs of a given type. The types that are present in LogiX are similar to the ones used in any other programming language and include bools (true/false values), floats (decimal point numbers), ints (whole numbers), strings (text values) and impulses. The impulse is a unique type, which is used to simulate the flow of a program.

Apart from selecting and creating the LogiX nodes, the LogiX Tooltip is also used to link them together. By connecting the nodes, the output of one is used as the input of another, which allows creating meaningful programs. Linking the nodes is possible by placing the tip of the LogiX Tooltip into the output of one node and dragging it into the input of the same type of another node. This creates a visual wire that symbolizes the connection. An example of a LogiX program performing integer addition can be seen in the figure 3.12.

As can be seen in the figure 3.11, the LogiX nodes are divided into several categories, each with a clearly defined purpose. To have a clear idea of what is happening in the practical part of this thesis, the most commonly used node categories are introduced and explained.

- The *Input* category contains nodes used to define the values of a given type (float, int, string), which are then used as inputs of other blocks.
- The *Operators* category holds nodes that perform operations on given inputs and outputs the result. This includes number and string addition (concatenation), multiplication, division, comparison, type checking, and more.
- The *Slots* category allows performing slot related actions such as destroying or duplicating a slot, changing the slot's parent, activating or deactivating the slot, and more.
- The *Variables* category provides nodes capable of storing and retrieving values of a given type.
- The *Flow* category contains nodes that are used to control the flow of a program. An example of this is a conditional branching, looping, generating impulse upon a value change, and more.
- The *Actions* category holds nodes that are used to interact with variables. It allows performing actions such as incrementing a counter or setting the value of a variable.

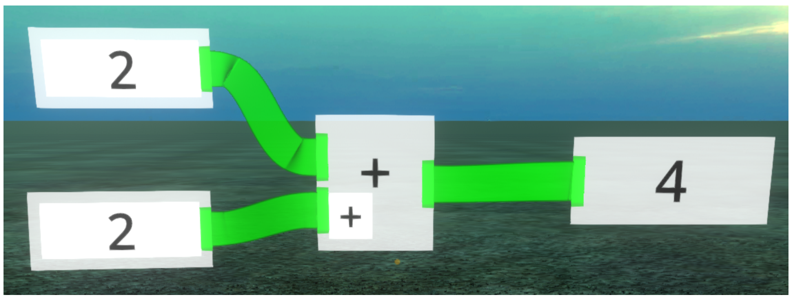


Figure 3.12: LogiX program composed of two int inputs, addition, and display

While it is nice to create LogiX programs similar to the one shown in the figure 3.12, it does not add to the interactivity of the virtual world. To fix this, the LogiX needs to be connected directly to the object's properties. This can be achieved by holding the grab button while aiming at the slot's (or component's) name in the inspector and pressing the secondary action button on the controller (or by left-clicking while holding shift in the screen mode). This creates a visual panel in the scene with all of the values of the given slot or component. These values can then be used as input or output for any LogiX nodes, creating an interactive object. An example of this can be seen in the figure 3.13, which shows a LogiX program concatenating two strings and displaying it using a *TextRenderer* component on a text object. [20]

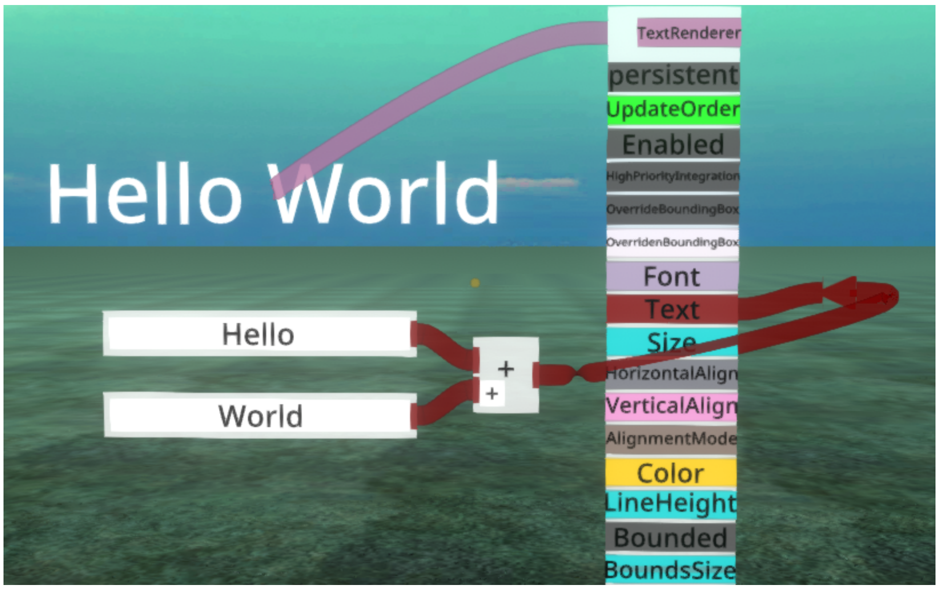


Figure 3.13: LogiX controlled text object

When working on a complicated LogiX program that contains more than a few nodes, it is often easy to get lost because LogiX does not provide any organization of blocks (such as functions in classical programming). However, this can be easily solved by using a community-created tool called Blueprints. This tool was created by the PolyLogiX group [21], which is well known in the world of NeosVR. The Blueprints can be found in the PolyLogiX folder inside the public section of the inventory. Upon spawning it into the scene, the Blueprint can be used as a magnetic board that holds and align any LogiX block attached to it. This way, it is possible to create smaller segments that operate together, creating a more organized program. Another useful feature that the Blueprints offer is to minimalize the board, hiding all attached LogiX blocks, without removing any functionality. An example of a LogiX program using the PolyLogiX's Blueprint can be seen in the figure 3.14. [10]

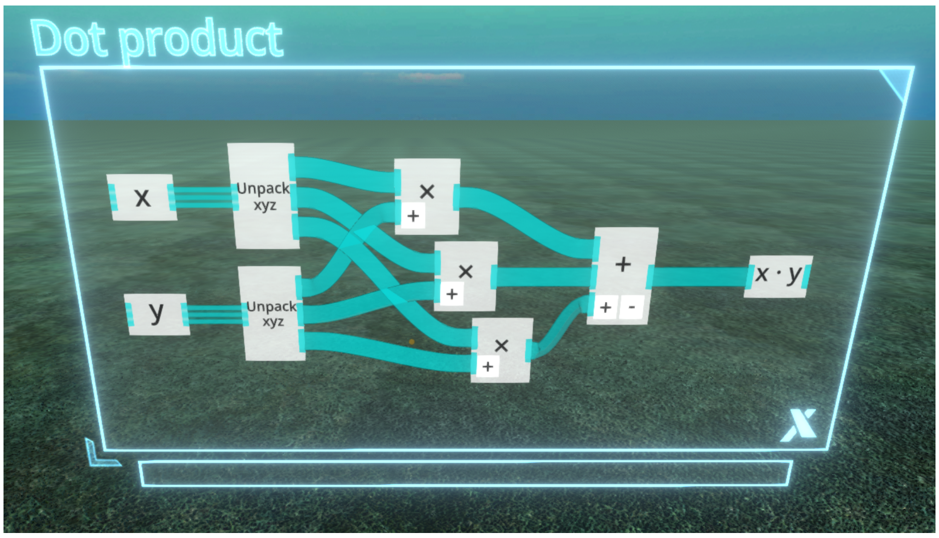


Figure 3.14: LogiX program computing dot product organized via Blueprint

The last thing that the LogiX Toopltip offers is the option to pack and unpack the nodes. Packing the nodes is important because it removes the visuals while still maintaining the functionality of the blocks. Packing can be performed by opening the controller menu and selecting the “Set Packing Root” option. Once the packing root is selected, the visuals can be removed by pointing the laser at the node and holding the secondary action button. To make the nodes appear back on the screen, the user should select the “Reveal LogiX nodes in children” option in the controller menu. [22]

Chapter 4

Designing the world

The previous chapters introduced the necessary theoretical concepts related to vector spaces and virtual reality. In this chapter, a world representing vector spaces is designed, which will provide a groundwork for the implementation.

To take full advantage of virtual reality, it is a good idea to create a world that looks and feels similar to the real one. Because of this, the environment of the virtual world is crucial. Apart from the correct setting, the world will need several objects to make it useful. Firstly, an object that will represent the vector spaces. Secondly, a tool, allowing the user to interact with the vector spaces. And lastly, an interactive board used to perform the operations with the vectors.

4.1 Environment

Because most of the users using this virtual world will be either students or teachers, the world will be set into a classroom-like environment.

4.2 Vector space

The vector space will be represented by a three-dimensional coordinate system. This coordinate system will use three axes (x, y, z) with a fixed range. This range should, however, be modifiable, allowing the user to stretch or shrink the space if necessary. The points of the coordinate space will be symbolized by small spheres with a label, displaying the coordinates of that given point.

The vectors will be represented by a line with a cone at the end to symbolize the direction. It will be possible to display vectors that both start and end at a point of the vector space, any other vector will be impossible to create. Similar to points, vectors will also have a label displaying the vector's components.

4.3 Vector tool

The vector tool will be a user-operated device allowing to interact with the elements of the vector space. This tool will have three different operation modes.

- *Create* mode, allowing the user to create new vectors.
- *Select* mode, allowing the user to transfer vectors from the vector space to the interactive board.
- *Delete* mode, allowing the user to delete vectors from the vector space.

The tool will also display the currently selected operation mode and change it upon pressing the secondary action button on the controller.

4.4 Interactive board

The interactive board will be the main control element of the world. It will contain three tabs, each related to different concepts of vector spaces.

The first tab will be about vector algebra, allowing the user to perform operations such as addition and subtraction of vectors, vector products, and others introduced in section 2.4. It will have space for up to two vectors and one editable number field to serve as the arguments of the operations. Each operation will have a separate button labeled with a symbol (+ for addition, − for subtraction, ...). These buttons will be clickable only if the required arguments are chosen, and it will be possible to have only one operation selected at a time. There will also be a button to show the geometrical representation of the result in the vector space (if possible).

The second tab will allow the user to perform matrix transformations (section 2.5). It will have space for a single vector and a matrix with editable number fields. It will also contain a button to execute the matrix-vector multiplication and a button to visualize the result in the vector space.

The third tab will study the collections of vectors and related concepts including, linear dependence (section 2.6), bases (section 2.7), and orthogonality (section 2.8). It will have a space for up to three vectors and will provide information related to this set. In case the selected set forms a basis, there will be an option to select one additional vector, which will be expressed by the coordinates.

Apart from the described aspects, each tab will contain a blackboard. This blackboard will provide the user with instructions on what to do and, in case any operation is selected, will display the result and the steps leading up to it. A wireframe showing the design of the interactive board with the *Vector algebra* tab selected can be seen in the figure 4.1.

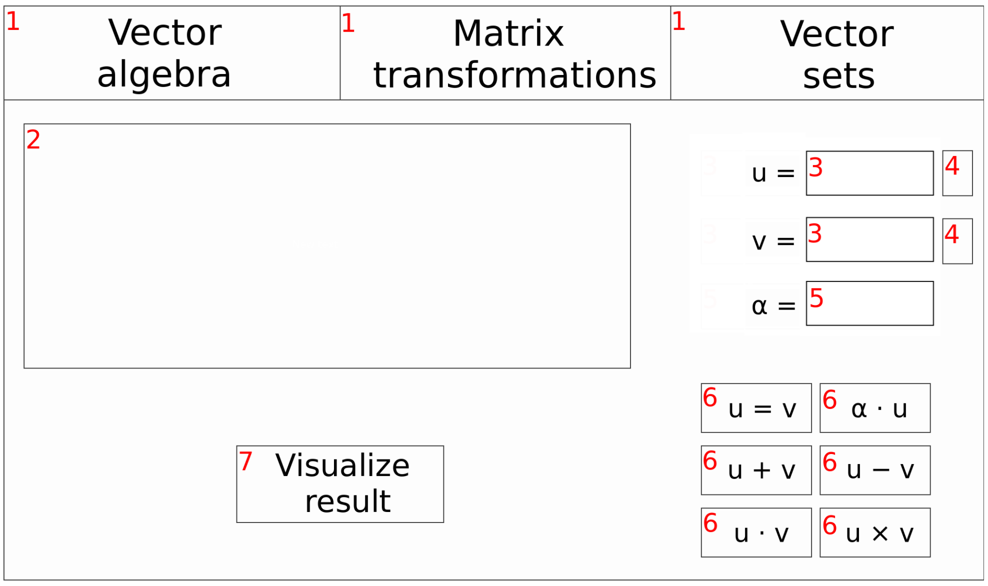


Figure 4.1: Wireframe of the Vector algebra tab

1. Tabs used to change the layout of the interactive board. Only one tab can be selected at a time, which is signaled by a differently colored background.

2. Blackboard, showing the instructions related to the currently selected tab or displaying the results and steps of the selected operation.

3. Vector fields filled by the *Vector tool* and labeled with the vector's components (if set) or with "?" (if not set).

4. Buttons used to remove the selected vector from the board.

5. User editable integer field accepting values from −9 to 9.

6. Buttons used to perform vector operations. Only one operation can be selected at a time, and the button should be selectable only if the required arguments are filled.

7. Button used to visualize the result of the currently selected operation. This button should be visible only if the result of the currently selected operation has a geometrical meaning and be hidden otherwise.

Chapter 5

Implementing the world

This chapter will concern the implementation of the virtual world representing vector spaces. The implementation will be done entirely in the NeosVR application and will follow the design draft proposed in the previous chapter.

The implementation will be done in multiple steps, each producing a part of the virtual world. At the end of this chapter, the individual components are put together, resulting in a fully functional virtual world.

5.1 Environment

The environment was created by spawning objects from the public inventory and placing them into the scene. Apart from purely esthetical models, a few message boards were also created to inform the user about how to use the world.

5.2 Vector space

The first interactive component of the virtual world is the representation of the vector space. This representation is split into two parts (points and vectors) that are implemented separately. Apart from the representation, the implementation of the vector space also included the creation of a control panel. This control panel can be used to modify the coordinates ranges or scale the vector space.

5.2.1 Points

The sphere model for the point was created by the ShapeTip tool, which took care of setting the required components (mesh, renderer, collider). The label was then created by programming a classical text object (created by the DeveloperTooltip) with LogiX. The blueprint of the LogiX program, which takes the position of the object and transforms it into a string representation, can be seen in the figure 5.1. This blueprint also produces a tag that is used to identify a specific point. The finished point representation can be seen in the figure 5.2.

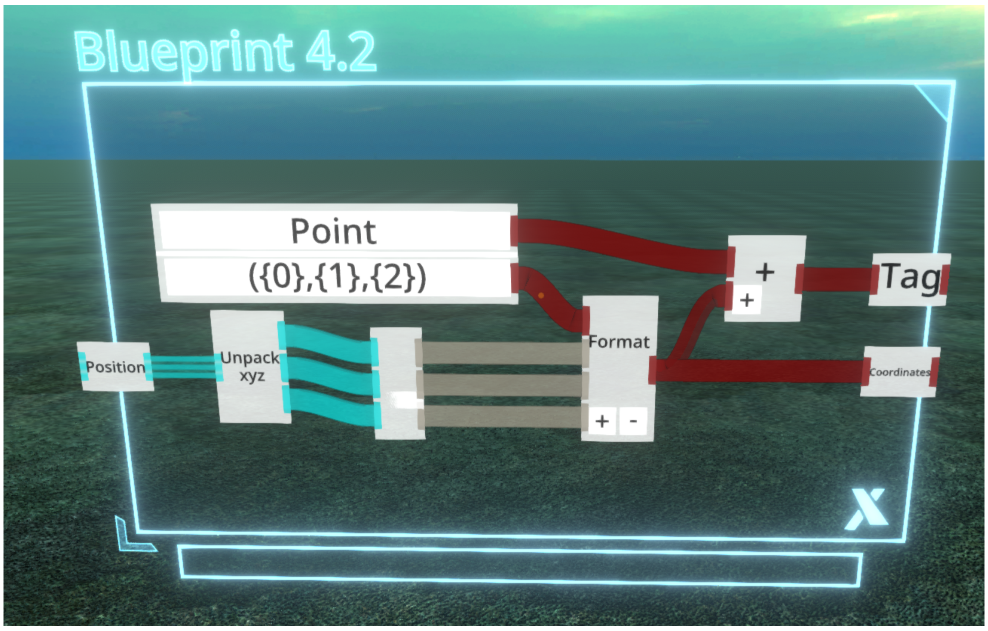


Figure 5.1: LogiX blueprint transforming position into a coordinates label

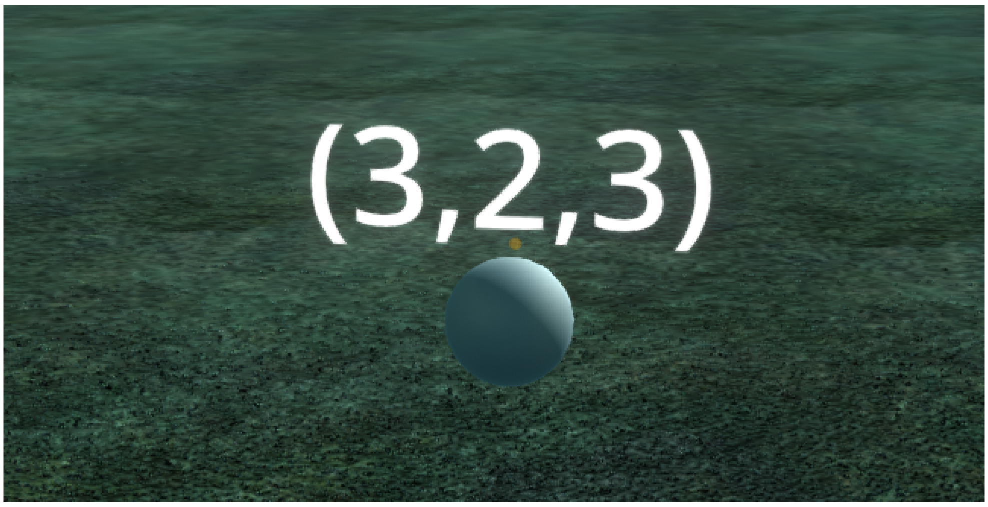


Figure 5.2: The point representation

5.2.2 Vectors

The implementation of the vectors was slightly more complicated because it needed to be done entirely from scratch. The model was made by using the *ArrowMesh* component. This component allows specifying a vector to render, which is ideal for this use case.

The other necessary parts of the vector are the collider (to click the vector) and the label (to identify the vector). Both of these need special aligning so that they copy the vector's model. This aligning is achieved by the *LineTransform* and *LineSegment* components, which both need two anchor points. These anchor points will be required during the creation of the vector and will ensure that the vector is positioned correctly in the world. The LogiX blueprint responsible for setting the anchor points and calculating the resulting vector can be seen in the figure 5.3.

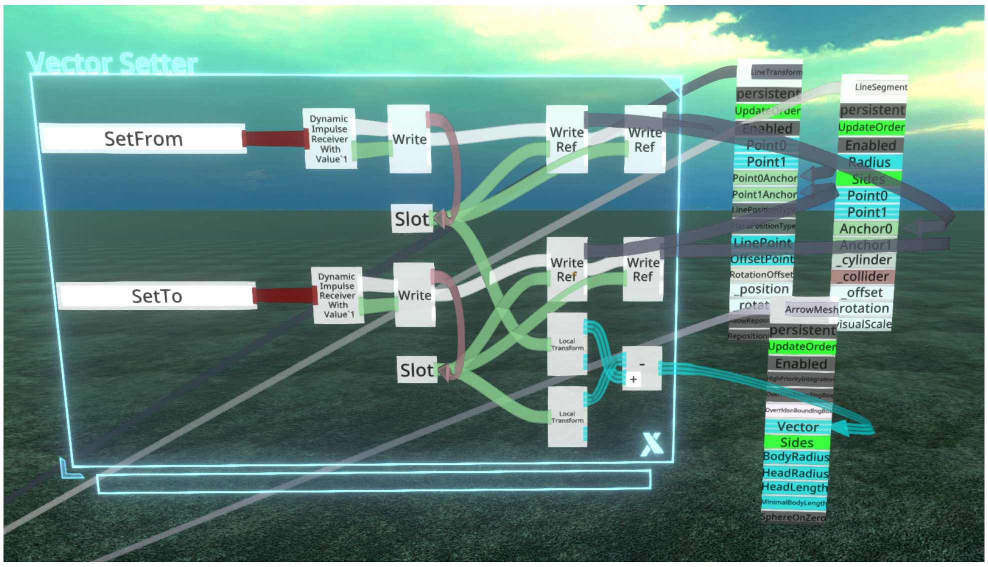


Figure 5.3: LogiX blueprint setting the vector's anchors

The above figure also shows the usage of *Dynamic Impulse Receivers*. These receivers are used in places where it is not possible to link the LogiX nodes directly. The reason for this is that the vectors will be created by aligning a copy of the template object. The way this works will be shown later during the implementation of the vector tool.

Apart from setting the vector, the vector implementation contains a few other LogiX programs. These programs are all quite simple and are used to set the correct text on the label or change the color of the vector. An example of several vectors placed inside the vector space can be seen in the figure 5.4.

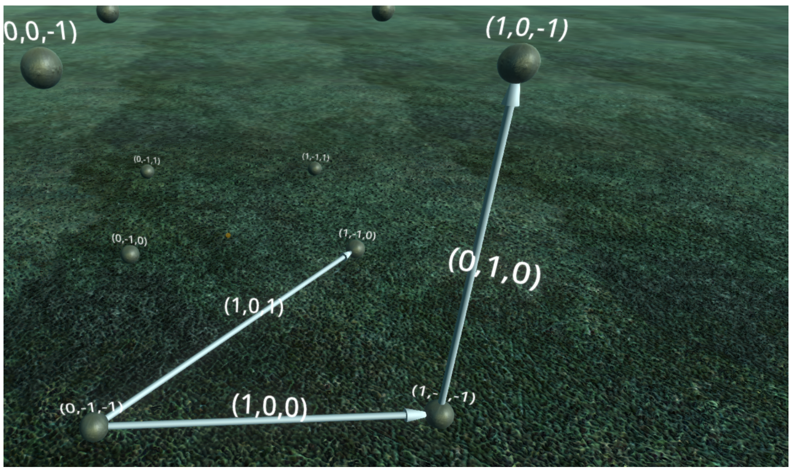


Figure 5.4: The vector representation

5.3 Vector tool

The core of the vector tool was created according to [23], which produced an equippable tool. This tool did not have any special features but allowed the user to aim at objects and obtain their slots by pressing the trigger button.

The next step was to create the mode swapping ability. This was achieved by placing a LogiX programmed text label above the tool. The blueprint with this program can be seen in the figure 5.5 and fires every time the secondary button is pressed. One thing that is interesting about it is the *Multiplex* node. This node takes several inputs and returns only one based on a given index, which makes it ideal for places that have a predefined number of possible values.

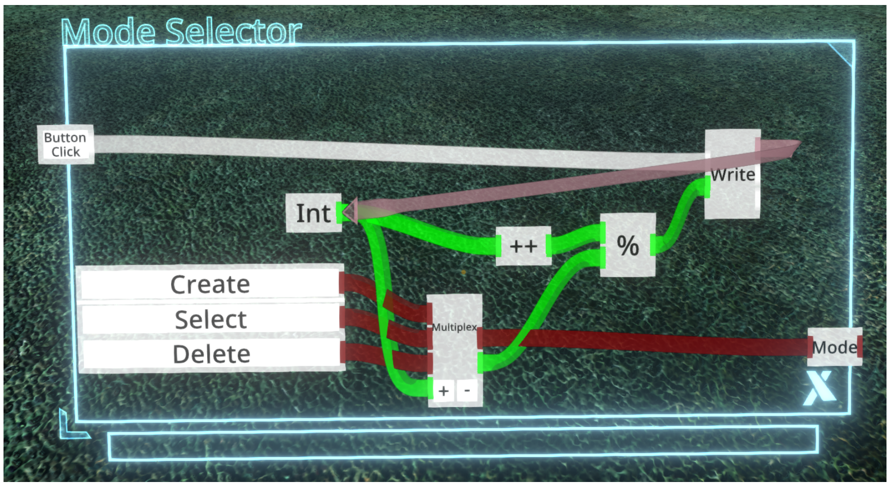


Figure 5.5: LogiX blueprint switching the tool's modes

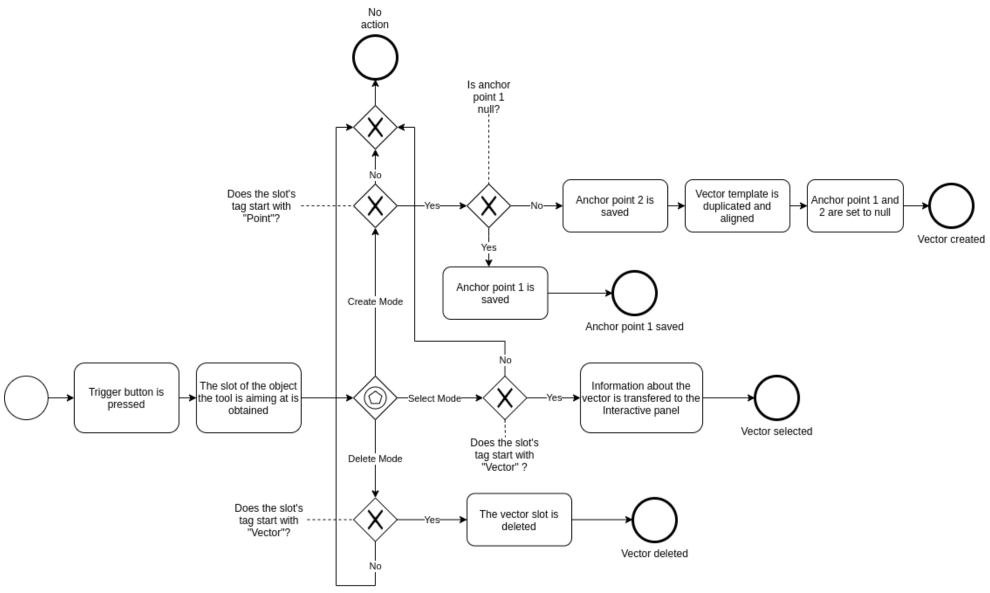

No action
Is anchor point 1 null?
Does the slot's tag start with "Point"?
No
Yes
No
Anchor point 2 is saved
Vector template is duplicated and aligned
Anchor point 1 and 2 are set to null
Vector created
Yes
Anchor point 1 is saved
Anchor point 1 saved
Create Mode
No
Trigger button is pressed
The slot of the object the tool is aiming at is obtained
Select Mode
Yes
Information about the vector is transfered to the Interactive panel
Vector selected
Does the slot's tag start with "Vector" ?
Delete Mode
Does the slot's tag start with "Vector"?
Yes
The vector slot is deleted
Vector deleted
No

Figure 5.6: BPMN diagram of Vector Tool flow

The subsequent steps included the implementation of LogiX programs responsible for performing the actions based on the selected mode. The flow of this process is symbolized by the BPMN diagram shown in the figure 5.6. In this diagram, each rectangle represents a whole LogiX program in NeosVR, from which most are trivial. The only program that is worth mentioning is the one responsible for the creation of the new vector (shown in the figure 5.7) because it uses *Dynamic Impulse Triggers*. These impulses are received in the vector and are handled by the *Vector Setter* program previously shown in the figure 5.3. The finished tool with the *Select* mode active can be seen in the figure 5.8.

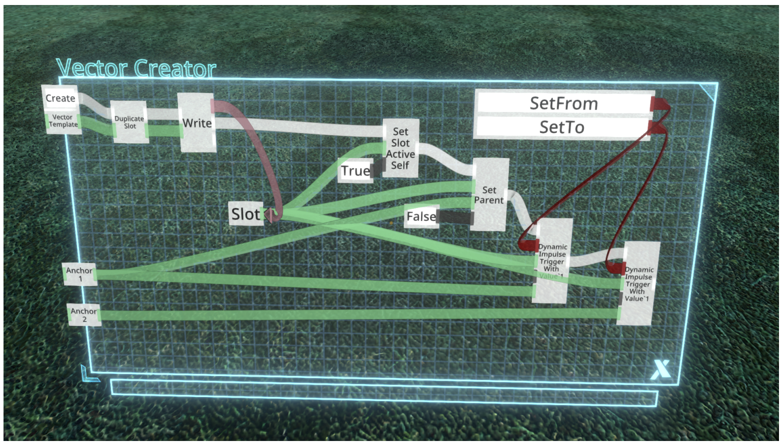


Figure 5.7: LogiX blueprint creating a new vector

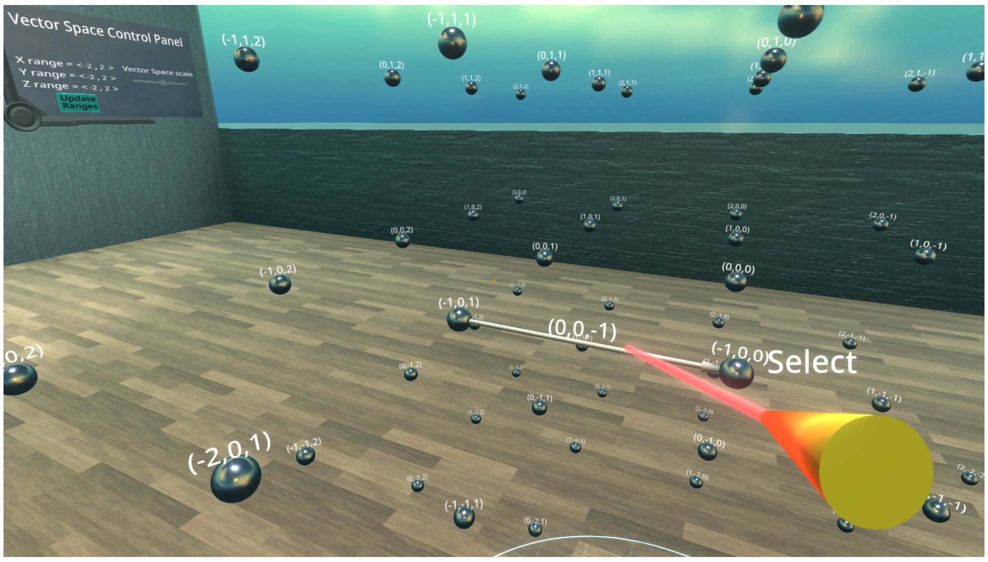


Figure 5.8: Using Vector tool to select vector

5.4 Interactive Board

The interactive board features three independent layouts, each providing access to a different set of operations and information. The layout can be switched by selecting a different tab from the panel at the top of the board. This was implemented by having three objects (each containing the whole layout) that are disabled by default and enabled only if the appropriate tab is selected.

Each layout contains a blackboard and fields for one to three vectors. These vectors can be filled by using the vector tool with select mode. Next to each vector is an "X" button, which can be used to remove the selected vector from the board. These buttons (and any other buttons used on the board) were implemented by using the *TouchableData* component, which allows obtaining click and hover events. The rest of the layout is specific and varies based on the selected tab.

The *Vector algebra* tab contains two fillable vector fields and an editable number field. It also features six buttons that are used to perform various vector operations. These operations are vector equality, scalar multiplication, vector addition, vector subtraction, dot product, and cross product. Upon pressing each button, the result and the steps leading up to it are shown on the blackboard. If the result of the selected operation is a vector, another button is shown that allows visualizing the result in the vector space. The vector algebra tab performing the vector addition operation can be seed in the figure 5.9.

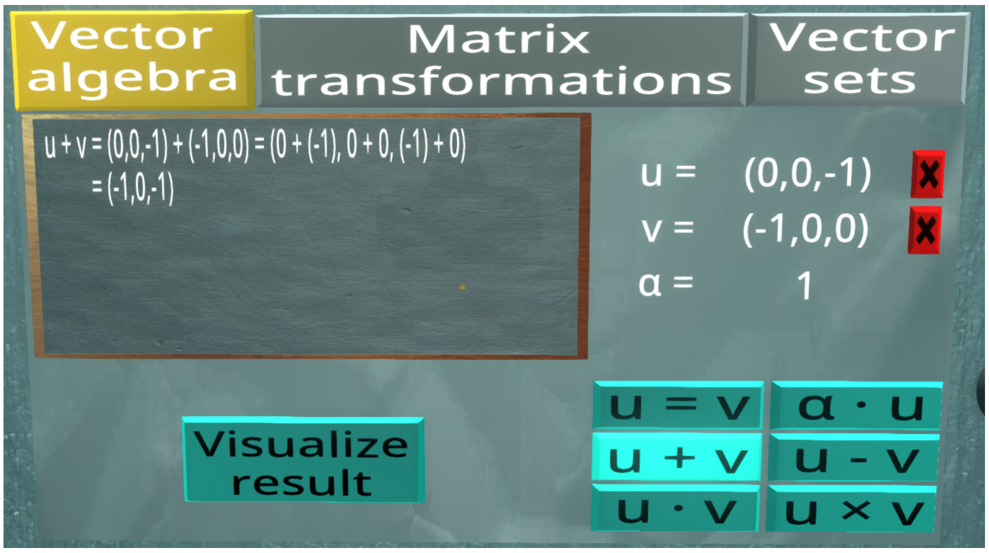
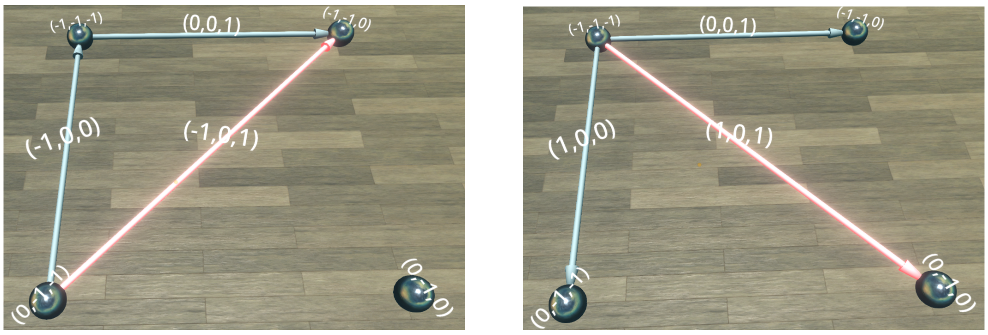


Figure 5.9: Interactive board with the vector algebra tab selected

The *Matrix transformations* tab contains one fillable vector field and a 4×4 matrix. The matrix can be modified either by editing its elements one by one or by using the transformation matrix creator at the bottom part of the tab. This creator allows generating the matrix by specifying the translate, scale, and rotate arguments. Similarly to the vector algebra tab, the steps of the multiplication are shown on the blackboard, and the result can be visualized.

The *Vector sets* tab contains three fillable vector fields that together form a vector set. The properties of this set are then displayed on the blackboard and contain the information about whether the given set is orthogonal, linearly independent, and if it forms a basis or not. In case it does, another vector field is shown, which can be used to express the vector with the coordinates relative to the selected set.

Each layout is operated by multiple LogiX programs that are using the same principles as the previously shown objects (points, vectors, vector tool). An interesting thing is the result visualization, which creates a new vector. The way this works is the same as in the vector tool (shown in the figure 5.7), but the anchor points are obtained differently. The starting point of the result is chosen based on the operation and its arguments so that the representation holds the best geometrical significance. The ending point of the vector is then calculated based on the starting point and located in the vector space by the slot's tag. An example of different visualizations based on the operation's arguments can be seen in the figure 5.10.



(a) Vectors with common point (b) Vectors with common origin

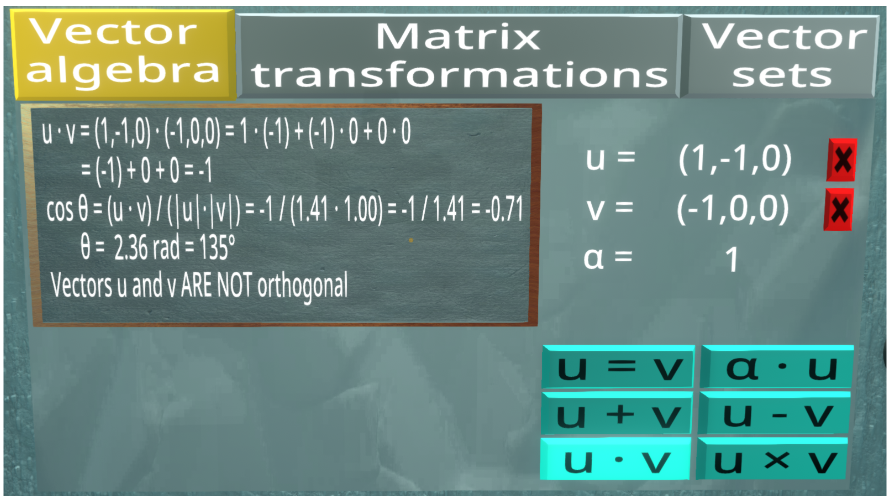
Figure 5.10: Different geometrical representations of vector addition

5.4.1 Examples of use

Each layout of the interactive board provides the user with multiple actions, which he can perform. Presented here is a single example for each tab so that the reader can catch a glimpse of what can be done in the virtual world.

Vector algebra

This tab is used to perform basic vector operations and (if possible) visualize the result. An example of calculating a dot product of two vectors u and v can be seen in the figure 5.11.



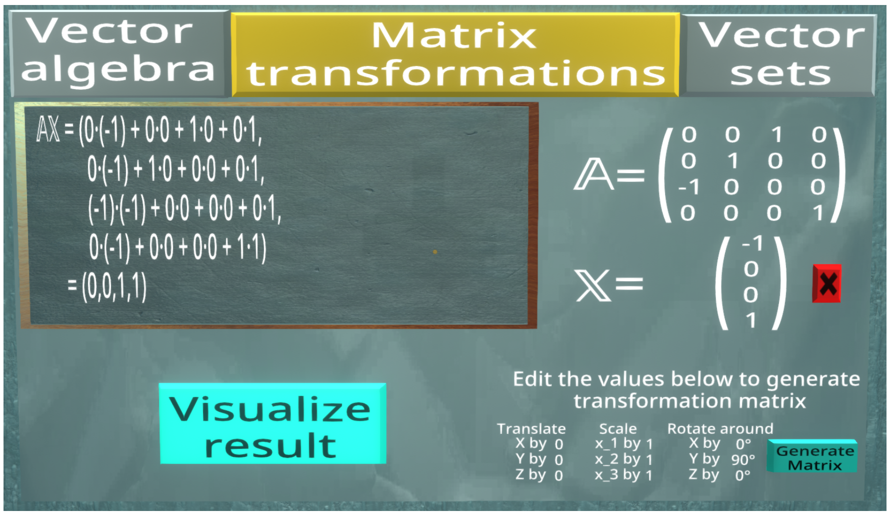
u · v = (1,-1,0) · (-1,0,0) = 1 · (-1) + (-1) · 0 + 0 · 0
= (-1) + 0 + 0 = -1
cos θ = (u · v) / (|u| · |v|) = -1 / (1.41 · 1.00) = -1 / 1.41 = -0.71
θ = 2.36 rad = 135°
Vectors u and v ARE NOT orthogonal

u = (1,-1,0)
v = (-1,0,0)
α = 1

Figure 5.11: Calculating a dot product of two vectors

Matrix transformations

This tab is used to transform the vectors by using a matrix-vector multiplication. An example of rotating the vector around the y-axis by 90° can be seen, along with the visualization, in figures 5.12 and 5.13.



$$\mathbb{A}\mathbb{X} = (0\cdot(-1)+0\cdot 0+1\cdot 0+0\cdot 1,\ 0\cdot(-1)+1\cdot 0+0\cdot 0+0\cdot 1,\ (-1)\cdot(-1)+0\cdot 0+0\cdot 0+0\cdot 1,\ 0\cdot(-1)+0\cdot 0+0\cdot 0+1\cdot 1) = (0,0,1,1)$$

$$\mathbb{A}=\begin{pmatrix} 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$\mathbb{X}=\begin{pmatrix} -1 \\ 0 \\ 0 \\ 1 \end{pmatrix}$$

Edit the values below to generate transformation matrix

Translate	Scale	Rotate around
X by 0	x_1 by 1	X by 0°
Y by 0	x_2 by 1	Y by 90°
Z by 0	x_3 by 1	Z by 0°

Figure 5.12: Rotating vector around the y-axis by 90°

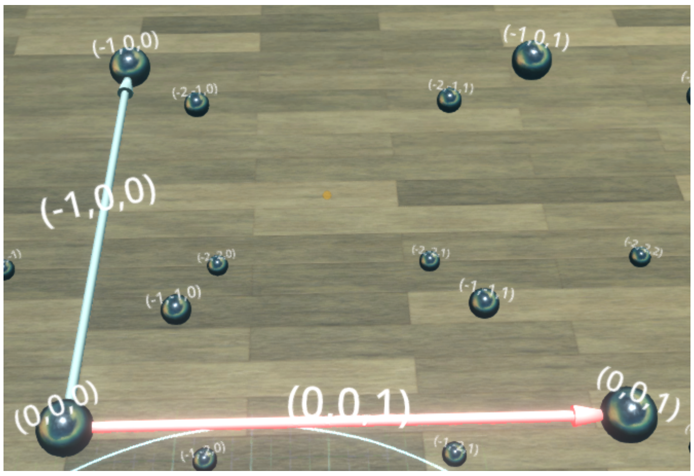
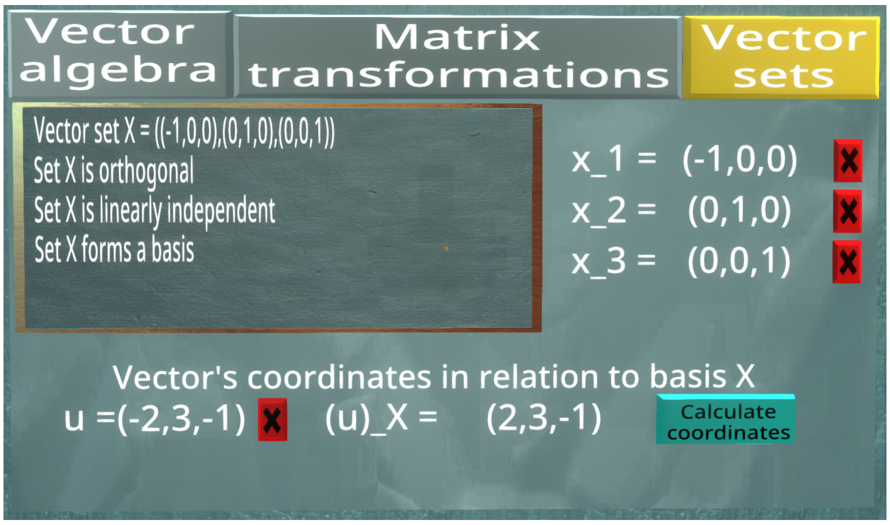


Figure 5.13: Visualization of vector rotation around the y-axis by 90°

Vector sets

This tab is used to study the properties of whole sets of vectors. An example showcasing a vector set forming a basis and expressing a different vector by coordinates relative to that basis can be seen in the figure 5.14.



Vector set X = ((-1,0,0),(0,1,0),(0,0,1))
Set X is orthogonal
Set X is linearly independent
Set X forms a basis

x_1 = (-1,0,0)
x_2 = (0,1,0)
x_3 = (0,0,1)

Vector's coordinates in relation to basis X
u =(-2,3,-1) (u)_X = (2,3,-1)

Figure 5.14: Expressing vector by coordinates relative to basis X

5.5 Finishing touches

The last step of the implementation was to put everything together. This was achieved by positioning the interactive objects in the environment and by interconnecting the individual LogiX programs. Afterward, all LogiX nodes were packed, and this thesis was inserted into a document reader inside the world. Finally, the world was published in the community hub under the name *Vector Spaces*. A scene from the finished world can be seen in the figure 5.15.

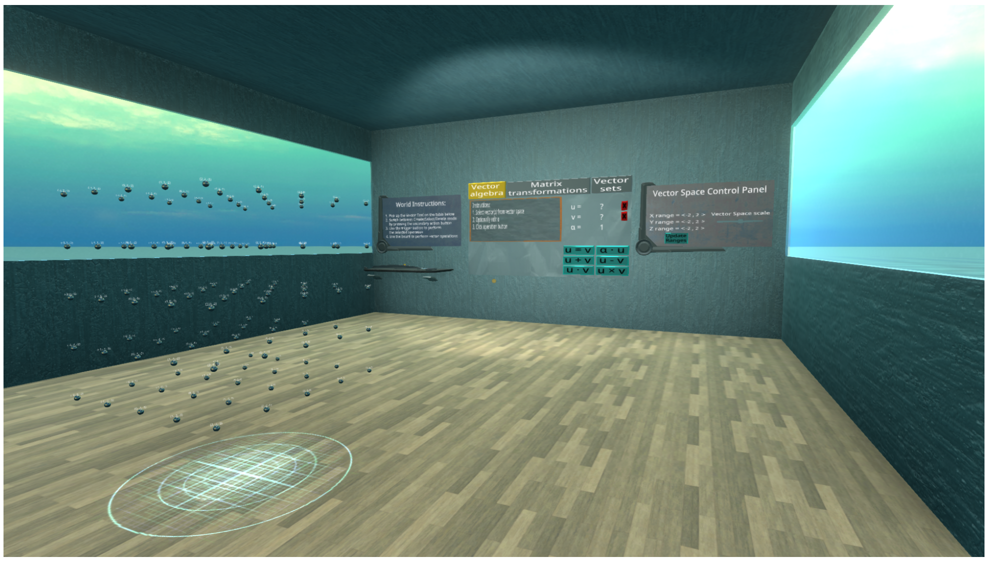


Figure 5.15: Scene showcasing the finished world

Conclusion

The main objective was to create an educational virtual world, which would help students to understand concepts of vectors and vector spaces. This objective was achieved by using the NeosVR application to create and configure the world, and visual programming language LogiX to program the individual components.

The resulting world can be used to create vectors in 3D coordinate space and visualize three main concepts of vector spaces: vector algebra, matrix transformations, and vector sets. The world was also published to the community hub and can be accessed by anyone through the NeosVR application.

In the future, the world can be improved by adding new, more complex vector transformations, and by visualizing other concepts related to vector spaces.

Bibliography

1. HALMOS, Paul R. *Finite-dimensional vector spaces.* 2. ed. Princeton: Van Nostrand, 1958. ISBN (hardcover).
2. DOMBEK, Daniel; KALVODA, Tomáš; KLEPRLÍK, Luděk; KLOUDA, Karel. *Lineární algebra* [online]. Praha: KAM FIT ČVUT, 2019/2020 [visited on 2020-04-15]. Available from: `https://kam.fit.cvut.cz/deploy/bi-lin//lin-text.pdf`.
3. BEEZER, Robert A. *Vector Spaces* [online]. 2015 [visited on 2020-04-06]. Available from: `http://linear.ups.edu/html/section-VS.html`.
4. LASS, Harry. The algebra of vectors. In: *Vector and tensor analysis.* New York: McGraw–Hill, 1950, pp. 1–28. ISBN 9780070365209.
5. *Cross product parallelogram* [online]. Wikimedia Foundation, 2020 [visited on 2020-04-09]. Available from: `https://commons.wikimedia.org/wiki/File:Cross_product_parallelogram.svg`.
6. HRABÁK, Pavel. *Handbook to BIE-LIN: 2. Systems of Linear Equations* [online]. 2020 [visited on 2020-04-16]. Available from: `https://courses.fit.cvut.cz/BIE-LIN/topics/handbook/bie-lin-handbook-02.pdf`.
7. HRABÁK, Pavel. *Handbook to BIE-LIN: 7. Determinant* [online]. 2020 [visited on 2020-05-01]. Available from: `https://courses.fit.cvut.cz/BIE-LIN/topics/handbook/bie-lin-handbook-07.pdf`.
8. HRABÁK, Pavel. *Handbook to BIE-LIN: 3. Vector Spaces and Related Terms* [online]. 2020 [visited on 2020-04-15]. Available from: `https://courses.fit.cvut.cz/BIE-LIN/topics/handbook/bie-lin-handbook-03.pdf`.
9. BRONSON, Richard. *Matrix Methods: An Introduction.* 3. ed. Elsevier Science, 2014. ISBN 9781483216614. Available also from: `https://books.google.cz/books?id=FAHjBQAAQBAJ`.

10. KLÁN, Petr; MARIANČÍK, Tomáš. *Jak stavět virtuální světy v metaverzu Neos*. London: Solirax Ltd, 2019. ISBN 9788088320265.

11. *Oculus Rift Vr Headset* [online]. Shenzhen BestAI Internet Co., Ltd ., 2019 [visited on 2020-05-04]. Available from: `https://www.kindpng.com/imgv/hTobRR_oculus-rift-virtual-reality-headset-htc-vive-oculus/%5C#gal_oculus-rift-virtual-reality-headset-htc-vive-oculus-oculus-rift-vr-headset_hTobRR_316099.png`.

12. *Real VR Fishing* [online]. Facebook Technologies, LLC., 2020 [visited on 2020-05-04]. Available from: `https://www.oculus.com/experiences/quest/2582932495064035/`.

13. *Virtual Reality and Education* [online]. 2017 [visited on 2020-05-04]. Available from: `https://www.vrs.org.uk/virtual-reality-education/`.

14. *Carolina Biological VR Dissection Library* [online]. 2020 [visited on 2020-05-04]. Available from: `https://www.victoryxr.com/victoryxr-products/carolina-biological-animal-dissection-virtual-reality-solutions/`.

15. *Unity* [online]. Unity Technologies, 2020 [visited on 2020-05-04]. Available from: `https://unity.com/`.

16. *Unreal Engine* [online]. Epic Games, Inc., 2020 [visited on 2020-05-04]. Available from: `https://www.unrealengine.com/`.

17. *NeosVR Metaverse* [online]. Solirax Ltd, 2019 [visited on 2020-05-04]. Available from: `https://neosvr.com/`.

18. COLGAN, Alex. *Unity Core Assets 101: How to Start Building Your VR Project* [online] [visited on 2020-05-04]. Available from: `http://blog.leapmotion.com/unity-core-assets-101-start-building-vr-project/`.

19. *Steam* [online]. Valve Corporation, 2020 [visited on 2020-05-05]. Available from: `https://store.steampowered.com/`.

20. FROOXIUS. *Visual scripting in VR: #4 Controlling properties* [online] [visited on 2020-05-08]. Available from: `https://www.youtube.com/watch?v=toszxzp0pXw`.

21. *PolyLogixVR* [online] [visited on 2020-05-10]. Available from: `https://twitter.com/polylogixvr`.

22. FROOXIUS. *Visual scripting in VR: #7 Finalizing Your Setup* [online] [visited on 2020-05-10]. Available from: `https://www.youtube.com/watch?v=U68hjR43noI`.

23. PROBABLEPRIME. *Neos VR: BasicTooltip with RayCastOne* [online] [visited on 2020-05-19]. Available from: `https://www.youtube.com/watch?v=FwNwneCtav0`.

Appendix **A**

Acronyms

2D	2 Dimensions
3D	3 Dimensions
BPMN	Business Process Model and Notation
$\mathbb{C}$	Complex numbers
$\mathbb{N}$	Natural numbers
$\mathbb{Q}$	Rational numbers
$\mathbb{R}$	Real numbers
VP	Vector Space
VR	Virtual Reality

Appendix B

Contents of enclosed CD

```
readme.txt ....................... the file with CD contents description
src.......................................the directory of source codes
  world .................... the directory containing world related files
    readme.txt ............ the file describing how to access the world
  thesis.............. the directory of LaTeX source codes of the thesis
text ........................................... the thesis text directory
  thesis.pdf........................... the thesis text in PDF format
```