How to Build

VIRTUAL WORLDS IN METAVERSE NEOS



Petr Klán, Tomáš Mariančík

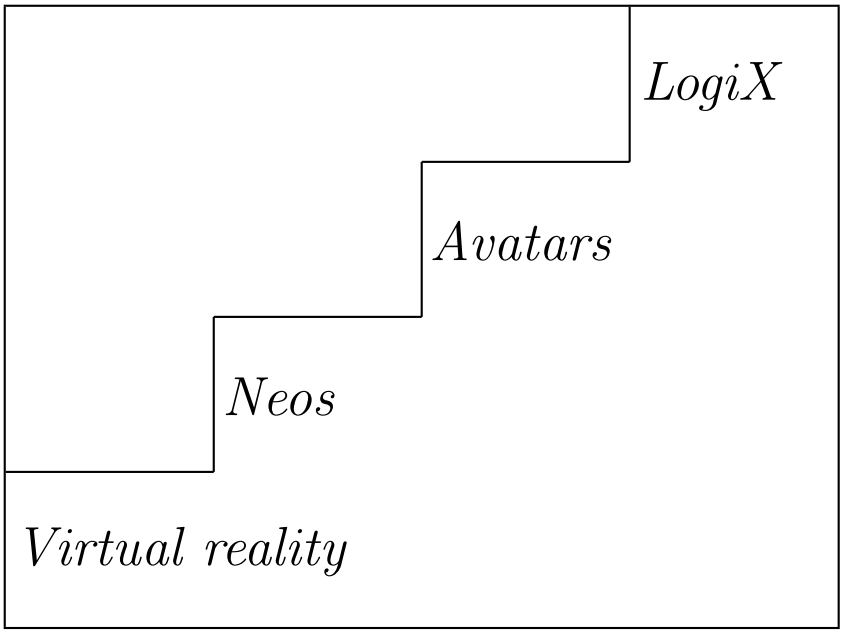
ISBN 978-80-11-02566-3

How to Build

Virtual Worlds in Metaverse Neos

How to Build
VIRTUAL WORLDS IN METAVERSE NEOS

Petr Klán, Tomáš Mariančík



Solirax Ltd London

Authors:
Petr Klán – University of South Bohemia in České Budějovice
Tomáš Mariančík – Solirax Ltd London

Reviewers:
Martin Kotek – Czech institute of informatics, robotics and cybernetics in Prague
Petr Pavlíček – First biology education in virtual reality, Opava

© Solirax Ltd London

ISBN 978-80-11-02566-3

Third edition December 2022

This book was prepared in typesetting system LaTeX

Contents

Preface . . . 9

1 VIRTUAL REALITY **11**
1.1 Thoughts of Virtual Reality . . . 12
1.2 Virtual Reality Devices . . . 15
1.3 Application Fields of Virtual Reality . . . 18
1.4 On Virtual Reality and Real World Relationship . . . 21

2 METAVERSE NEOS **25**
2.1 Download and Install . . . 26
2.2 Introductory World . . . 27
2.3 Virtual Cathedral Hub . . . 30
2.4 Empty World . . . 31
2.5 First Steps in New Worlds . . . 32
2.6 Gallery of Tooltips . . . 34
2.7 Best practices of building virtual worlds . . . 37
2.8 Interior and exterior adjustments . . . 40
2.9 Positioning of objects in scenes . . . 41
2.10 Transformation of objects . . . 44
2.11 Mathematics of transformations . . . 47
2.12 3D models, materials, textures and shaders . . . 48
2.13 Import of 3D models . . . 52
2.14 Terrains . . . 54
2.15 Particle Systems . . . 55
2.16 Lighting, photography and streaming scenes . . . 59
2.17 Scene inspector . . . 62
2.18 New objects . . . 64
2.19 Customized materials . . . 66
2.20 Saving and exporting objects . . . 71

3 AVATARS **73**
3.1 Social relationships and experiences of avatars . . . 74
3.2 Appearance, initialization and reincarnation of avatars . . . 76
3.3 How to scan an avatar . . . 80
3.4 Movements and animations of avatars . . . 81

4 VISUAL PROGRAMMING **85**
4.1 Visual Coding Principle . . . 86
4.2 LogiX Tool . . . 87
4.3 Counting with numbers, vectors and strings . . . 88
4.4 Working with time . . . 92
4.5 Color interaction . . . 94
4.6 Object programming . . . 95
4.7 Complex Scenes . . . 99
REFERENCES . . . 108

Preface

A virtual 3D world can be built from the proposed four components.

Virtual reality (VR) will be a pivotal 21st century technology, a new medium with the potential to change everything around education, leisure, work, creation, and more. Working in virtual reality will define the next generation of jobs and inspires us to broaden our creative horizons.

This book provides a toolkit for working in virtual environments, the objective being to build virtual worlds with the Neos metaverse engine (derived from the words *meta* and *universe*). Neos is free software which can be downloaded from the Steam platform. The book's content is concerned with elementary VR exercises which will teach the key ideas and systems of virtual environments.

Our aim is to provide an understanding of the principles and applications of working in VR, sufficient to build worlds for a wide variety of purposes. The authors believe this book will equip creative readers for tomorrow's workplace, provide ideas for integrating 3D virtual worlds to everyday 21st century life, as well as allowing them to push the bounds of imagination.

The first chapter *Virtual Reality* introduces virtual technologies and foundational 3D modeling and simulation concepts. The second chapter *Metaverse Neos* (*Neo* denotes new and *S* denotes space – altogether *new space*) describes how to operate within the metaverse itself. This presents techniques for generating content in virtual 3D environments, virtual scenes, interactions, etc. The third chapter *Avatars* deals with how humans present themselves and socialize in VR. In the fourth chapter we discuss the Neos visual programming language *LogiX*. It is primarily LogiX that which enables the dynamic behaviour and object interactions necessary to make virtual environments feel compelling and believable.

Beyond basic computer skills, no specific background knowledge is required to learn to create and build virtual worlds. However, since VR environments are mathematical objects at their core, knowledge of the underlying mathematics will be helpful, especially if the goal is to make virtual worlds mimic real ones, or to produce even more complex and imaginative constructs. Experience working with game engines, computer graphics, vector spaces, 3D models etc. will also transfer well.

VR development in the Neos metaverse can be done with or without a virtual reality headset; an ordinary computer screen is enough in many cases.

When you enter virtual space, you will probably be amazed by two phenomena:

1. A teleportation mechanism that allows avatars to instantly jump between worlds.

2. The spontaneity of collaboration supported by this world–hopping. While building in VR, it is very common to have human–controlled avatars visit you unexpectely, each of them bringing different experiences and knowledge from both virtual and real worlds. Such chance encounters sometimes lead to amazing spontaneous creative cooperation. Physically, the existence of teleportation is known at the quantum level. If there is spontaneous cooperation, as in the virtual world, then there may be a creative connection between human beings with the universe and with each other.

VR and its integration into real life is only just beginning. The rulebooks and guidelines for this new medium are still unwritten. Thus, any virtual space you build, either through

imagination or technological tools, can bring new insights into the language of virtual worlds and their interaction with the real one.

There are currently three major ways to create VR content:

1. Unity: an engine with a creative game editor to develop 3D content. This can be converted for use in VR `https://unity3d.com/unity`.

2. Unreal Engine: another engine with a complete set of creative tools to develop gaming and 3D projects, including VR programs `https://www.unrealengine.com`.

3. Neos: a metaverse with tools for virtual content production directly within VR `https://neosvr.com`.

Unity and Unreal Engine are primarily desktop applications. Virtual 3D content is created on ordinary 2D screens without using a headset display. A VR application is then generated at the end of this process. Neos, by contrast, is focused from the outset on building virtual content directly in VR. Therefore, it enables a much more efficient workflow for creating virtual 3D content which can be several times faster than traditional development methods.

The first course on the Neos metaverse and its possibilities can be found on the NeosVR Discord server (a discussion platform) or at the NeosVR channel `https://www.twitch.tv/neosvr` on the Twitch video streaming site.

The authors express their thanks to the reviewers for the comments and suggestions that led to the improvement of the book. The authors thank the Faculty of Information Technology of the Czech Technical University in Prague and Solirax Ltd London for their support.

Associate Professor Petr Klán, Ph.D. lectures on virtual reality and applied mathematics. Petr is concerned with applied maths and has published several research books, most recently titles *Process Control* (together with R. Gorez), *Numbers: Relationships, Insights and Eternal Inspirations*, *Modern Number Theory* or *Scientific Thinking*.

Tomáš Mariančík aka Frooxius is a co–founder, creator and designer of the start–up Solirax Ltd London, which develops the Neos metaverse. Tomáš won one of the grand prizes of the Intel ISEF Young Scientists and Engineers competition in Pittsburgh for innovative processor architectures and has created several VR games, including popular *SightLine* and *The Chair*. He is the author of the educational virtual application World of Comenius, which preceded the Neos metaverse.

The authors have a long history in common and were honored to work together on this book.

Each virtual scene used here was built and experimented with by the authors. Nevertheless, in spite of our best efforts, typos, errors or other inaccuracies may still be present. Should readers notice any issues, kindly notify us at the e–mail address: petr.klan@gmail.com. We thank you in advance.

Chapter 1

VIRTUAL REALITY

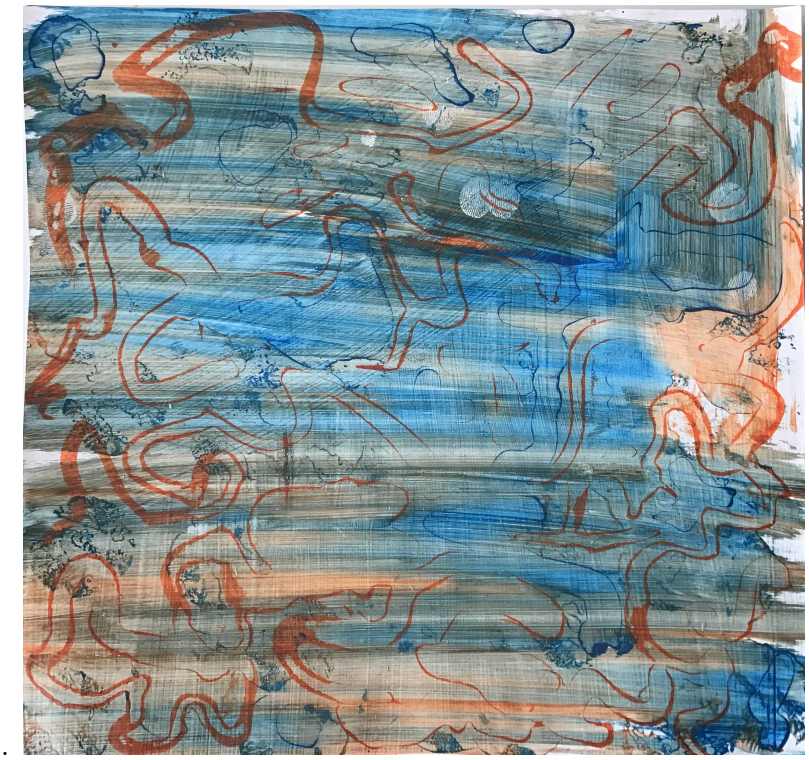


Figure 1.1: Virtual reality as an abstraction. Source: unknown author

The objective is to answer:

- How does virtual reality work?
- What devices does virtual reality use?
- What does virtual reality allow?
- How does virtual reality relate to the real world?

Fig. 1.1 illustrates a synthesis of the real world with immersive abstractions and ubiquitous imaginations of virtual environment entered in this chapter.

1.1 Thoughts of Virtual Reality

For implementation of virtual reality, you need specific glasses known as *headset* or *HMD* (Head Mounted Display). They either work wireless (stand–alone) or are wired to a computer using cable (tethered). The computer must be customized for virtual reality, which is usually known by terms *gaming* and *VR ready*. When you put glasses on the head and the glasses work, you see virtual environment more or less similar to the real environment you live in. Like the parts of the latter you naturally call *world* or *space*, it is in virtual reality talked about *virtual world* or *virtual space*.

Definition. *Virtual Reality* is a computer–generated 3D environment that can be experienced and influenced just like the real environment.

From the perspective of a true 3D environment, virtual reality is no different. It includes digitally generated virtual space, which includes virtual worlds. They are represented by the so–called *orbs* (according to Latin orbis – world). From a mathematical point of view, virtual reality is a pure mathematical construct. All properties, motions and interactions are done by number procedures that mathematicians have discovered, invented and simplified for centuries.

Virtual reality can be considered as a model of real environment or space, which can be simplified by features that are not important at the given moment. For example, a virtual tree may not be as large as real, you can't throw a virtual dice as real, or virtual rain may not form puddles or streams. On the other hand, virtual reality digitally represents space in which you can reside similarly to real space. That is why the word virtual is sometimes omitted and it is simply referred about space similarly as virtual or digital moneys are simply referred as moneys. Virtual things mean something that is not real, but when they work the same as real ones, like space, world or money, they are taken as real and the word virtual is not used.

Example 1.1 *Virtual environment of things.* Fig. 1.2 shows a virtual space with several things (objects) located in this space – two 360° photos, two so–called Klein bottles, enlarged coffee bean, mechanical Watt centrifugal regulator, conference poster, bumblebee photo, dashboard menu etc.

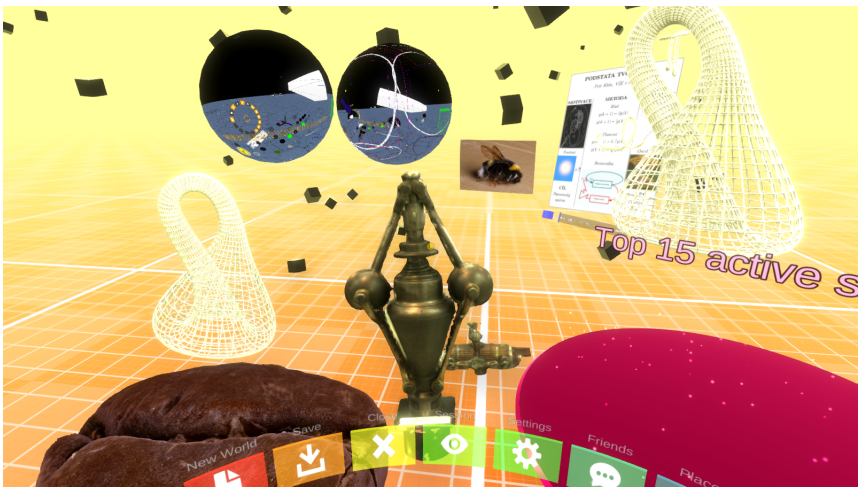


Figure 1.2: Part of virtual environment of things.

Example 1.2 *Virtual landscape.* Fig. 1.3 depicts a rainy, foggy and hilly virtual landscape with trees, house and wind power plant.

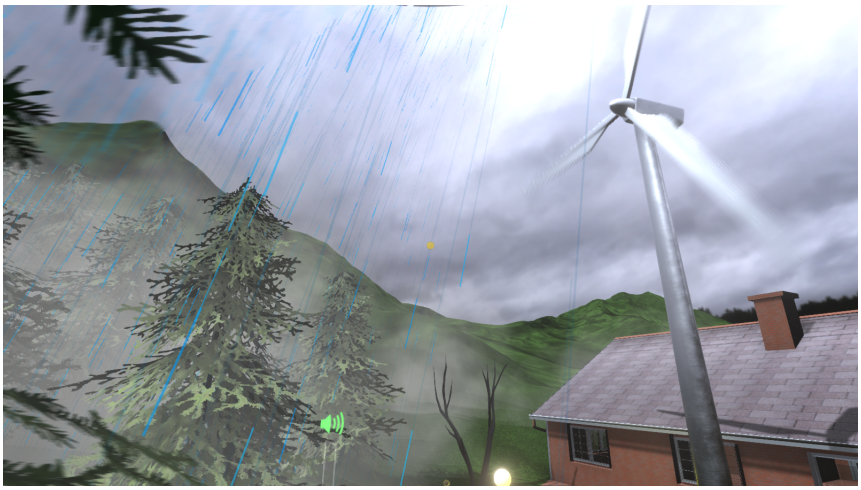


Figure 1.3: Virtual rainy, foggy and hilly landscape.

Example 1.3 *Virtual moon environment.* Fig. 1.4 presents a photogrammetrically reconstructed moon landscape after module Apollo's 14 landing. Earth is visible at the top left.

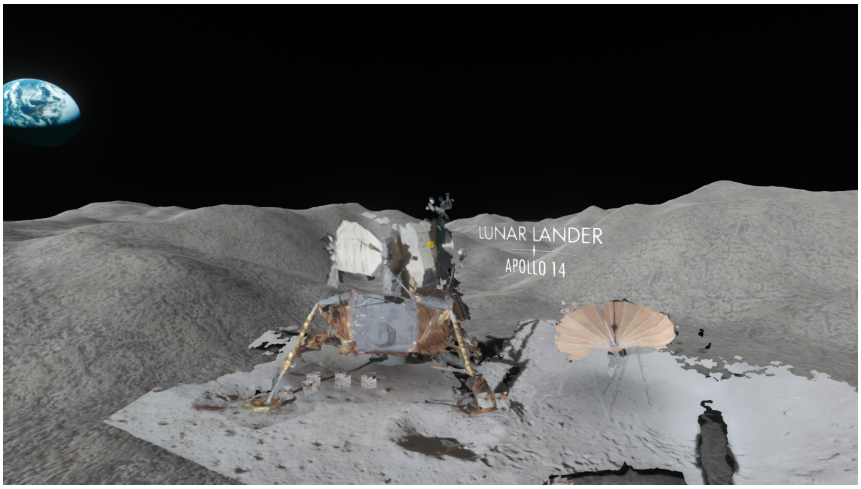


Figure 1.4: Moon landscape.

Example 1.4 *Virtual laboratory.* Fig. 1.5 shows a virtual development environment for visualization and control of an engine propeller turning if the LogiX visual programming is used (see Chapter 4).

Virtual environment is characterized by two important features that other media do not have: *immersion* and *presence*. Immersion refers to the ability to surround users with a virtual environment so that they can behave in it as much as in the real environment. Presence refers to the users feeling that they actually are in the virtual world, and that they are communicating and experiencing this environment as much as the real environment. Note that virtual reality is primarily created for the feature of presence. The latter is associated

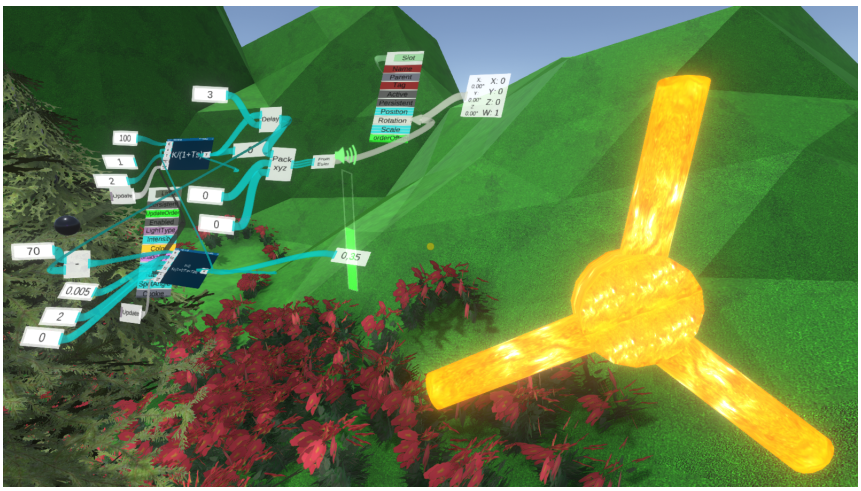


Figure 1.5: Development virtual space for visualization and control of an engine propeller turning.

with an optimal virtual application, and the immersion ability is a way to achieve it.

The effort to put specific glasses on the head and to see a copy of the real space has almost 200 years history. It includes different types from stereoscopes[1] (1849) for viewing images up to the latest wireless headsets Oculus Quest or Vive Cosmos (2019). Fig. 1.6 illustrates the changes in glasses over time. Until nearby now, virtual reality headsets have been technically demanding in use and economically inaccessible. A few years ago, the situation began to change dramatically towards greater applicability and economic availability of headsets. Furthermore, virtual reality has received support from both computers and their operating systems. The use of glasses has become much more comfortable.

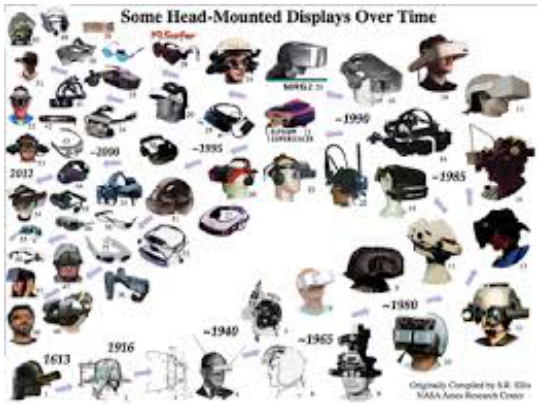


Figure 1.6: Changes in virtual glasses during time. Source: `https://ntrs.nasa.gov/archive/nasa/casi.ntrs.nasa.gov/20140011011.pdf`

In addition to virtual reality, you can meet the term *augmented reality* or *mixed reality*. Their relationship is schematically represented by the so–called virtual continuum, which is illustrated in Fig. 1.7. While virtual reality represents a complete virtual environment, augmented reality combines real and virtual elements so that virtual elements are incorporated to the real environment in more or less degree. Augmented reality can be achieved simply by turning off some elements of a virtual reality scene and replacing them by the real ones. For

[1] It was a kind of kaleidoscope. Kaleidoscopes were optical toys.

example, the sky will not be virtual but it can be replaced by the real one. Therefore, the authors prefer the virtual reality in this book and assume that individual elements of virtual scenes will be possible to turn off and replace with the real ones.

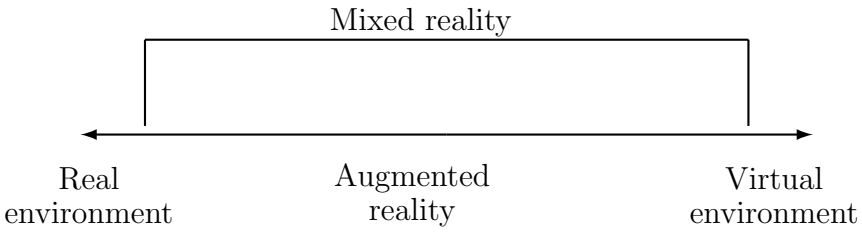


Figure 1.7: Virtual continuum: virtual and augmented reality relationship.

1.2 Virtual Reality Devices

The basic virtual reality set typically includes 1 headset, 2 handheld controllers and 2 devices to determine the current position in space, called *positioning stations*. Role of the positioning stations is to supply the data needed to calculate the current position of the headset in space.

In some types of headsets, positioning stations are integral part of the glasses. Headset's eyes (two, four, or more), which include built–in cameras, are a distinguishing sign. Current position of the headset in space is then calculated from images of the built–in cameras. Some types of glasses have positioning stations separately. They are intended to be placed either at opposite corners of the room where virtual reality is used or next to the computer to which the headset is connected. Area where the headset can really move is usually delimited when installing the headset software. A microphone and headphone, or at least a headphone jack, are a common part of glasses.

A processor that manages one or two displays operates in the hearth of each headset. It results in the stereoscopic effect that models human vision. To achieve this effect, the area of a single display is halved. Each part operates one eye. The headset includes a graphic card (in the case of wireless glasses) or it uses a graphics card of the connected computer (in the case of wired glasses). Some contemporary glasses, such as Oculus Quest, can work in both modes. In wireless mode, they use a built–in graphic card. If connected to a computer via cable, then the graphic card in computer is used to render image streams.

A human use of virtual reality presented by the authors iis n Fig. 1.8. Headset is mounted on the head. In each hand you can see one controller. The eyes on the headset indicate the inner position stations[2].

From a perspective of virtual reality use, the three components are important, as shown schematically in 1.9:

[2]As a general rule, when traveling or wearing the glasses more often, the authors use the ones that contain the positioning inherently. Otherwise, they usually choose glasses that have positioning stations separately, because they determine the current position more precisely and fit into larger spaces. For some types of headsets, a single pair of external positioning stations covers a bigger amount of headsets at the same time



Figure 1.8: Authors in virtual reality. Source: M. Kotek a K. Hulec

1. Glasses (headsets) that generate virtual environment.
2. Inputs that affect the virtual environment by the drivers.
3. Outputs, that transmit signals from the virtual environment. Nowadays, they are realized mostly by sounds. However, they can determine a haptic feedback too if using specially adapted controllers, gloves, etc. Note that outputs are the least sophisticated part of virtual reality now.

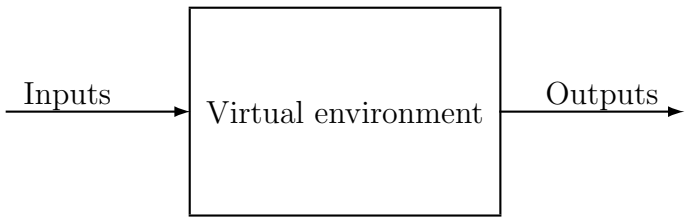


Figure 1.9: Virtual reality in as an inputs–outputs block.

Although different shapes of the controllers (for example, the Oculus controller is shown in Fig. 1.10), the way you control the virtual environment remains the same. Five control elements collected in Tab. 1.1 together with corresponding functions represents a standard.

Notice that you see both controllers in virtual scene. Each of them emits a laser–like beam. In order to realize a function, to snap and move an object of scene, for example, focus the beam on the object and press the side button with your hand.



Figure 1.10: Oculus controller.

Control element	Function	Position on controller	Control by
Trigger	Primary action	In front	Index finger
Big knob	Sekundary action	Upward	Thumb
Small knob	Controller menu	Upward	Thumb
Joystick	Locomotion	Upward	Thumb
Side button	Object snap	On side	Hand grip

Table 1.1: Knobs of controllers and their functions.

It is assumed that users will soon apply all human senses in the virtual environment. Virtual reality with a predominant visual and audio perception will not suffice to create an immersive experience, and a haptic feedback by physical stimuli will increasingly be used. The latter transfers from the virtual environment when, for example, you snap something in it.

The quality of the immersion depends on the sensitivity of virtual reality as well, ie on the difference between time you take action in the headset (for example, turning the head or pressing the knob of the controller) and when this change is visually reflected in the virtual world. This time is called *latency* and should be as small as possible, ideally not exceeding 20 ms. Latency is so important that when choosing between a higher latency/resolution headset and a lower latency/resolution headset, we prefer the second one[3].

Other accessories are important for the comfortable use of virtual reality as well. Here's an example of the basic assembly, including some practical use add–ons:

a) Computer (desktop or portable according to purpose) denoted as gaming and VR ready.

b) Headset including two controllers.

c) USB hub. A wired headset usually requires one HDMI and one USB connector. Some glasses need a USB–C connector only. Accessories associated with the headset may require additional USB connectors. Wireless headsets are powered by rechargeable batteries and charged via USB. It is therefore advisable to have enough USB connectors.

[3]The latency of the best current wired (tethered) headsets is 7 ms, which is equivalent to 144 frames per second (framerate). The latency of today's top wireless headsets is 13 ms, or 72 frames per second.

d) Rechargeable battery charger. Usually one or two AA batteries are used to power the controllers. Some controllers are based on built–in batteries. The latter will cover about 10 hours of operation. Therefore, it is wise to use rechargeable batteries for a frequent use.

e) USB–C to HDMI converter. If you plan to project the virtual environment on data projectors[4] and the headset requires an HDMI connector, then you need two HDMI connectors on the computer. In case of only one HDMI connector, the above converter can be used to make an additional HDMI.

f) 3D Object Scanner. If you plan to import real objects to the virtual environment, it is desirable to have a 3D scanner. A good compromise between quality, size and price is 3D scanner called *Structure Sensor* with built–in frame and cable (see Fig. 1.11) that connects to a tablet. This is a handy portable solution where you can easily bring the scanner to a real object and scan the object (even a human being) within a few minutes by slowly turning the object around with the scanner turned on. Tablet then processes the scanned object into .obj or other format, such as .fbx or .stl, etc. Related files could be sent (via e–mail for example) to a computer with glasses attached and then imported into a virtual environment.



Figure 1.11: Structure Sensor with iPad frame and interface cable.

1.3 Application Fields of Virtual Reality

Predictions indicate that by 2024, virtual reality will be as common as today's smart phones. It is expected that in a virtual environment, it will be possible to spend time in the same way as in the real world, whether it will be work, relaxation or entertainment.

The way of creating and building virtual worlds assumes the specific elements. Basic elements in current computers now are files that you create empty, fill with a content and save. Instead, basic element for virtual reality is the 3D world (*orb*), which you similarly create as empty with sky, earth and sun, you fill it with content and lsave. Primary activities that you can apply to fulfill virtual worlds are in the following list:

a) Use of tools or brushes to draw, shape models, apply color palettes etc.

[4]This is a good way to convey the content of the virtual environment to the audience without glasses.

b) Transformations of objects – shifts, scaling, rotations.

c) Import of objects.

d) Application of effects of particle systems such as rain, snow, flashes etc.

e) Changes of virtual object properties.

f) Creation of surfaces, materials and working with light.

g) Representation of human beings and their social experiences, creator jams etc.

h) Programming scenes and object interactions.

i) Movements – teleportation, flying, walking, etc.

j) Switch between different worlds and search for them.

k) Recording of virtual environment events – video streams, photos etc.

l) Export of virtual objects.

To achieve these goals, the authors will use the metaverse tool *Neos*[5]. In a virtual environment, it will be possible to work, pay, provide services, study, travel, shop, experiment and perform other activities just as in the real world. Neos allows to create and build virtual worlds focused on these purposes. It assumes a combination of imagination, visual programming, gaming skills and current computer habits used with files.

An enter world after the metaverse is started is shown in Fig. 1.12. It features a large screen displaying Neos video, and in the bottom right corner it is an orb of a world called *Neos Hub*. The latter presents a *cathedral* of worlds, ie a central place of experience and meeting where you can search in all submitted worlds or you can submit a world here. You enter the hub by focusing the beam of one controller on the orb and clicking the primary action with the index finger.

Virtual and mixed realities indicate *transformative technologies* that are likely to affect all business. Virtual users will apply ways of discovery, learning and communication that have never had a background. They will feel and connect with other users and places more intensively than ever before.

Virtual reality is at its beginning. What can the Neos metaverse do in virtual reality? Among other the following things:

a) Create games and play – board games in Neos Hub, virtual worlds The Cube, Chess etc.

b) Discover Earth and the Universe – Virtual Worlds Gravity Playground, Mars!, N - Relay 146, Planets!, Apollo Reconstructed Teaser etc.

c) Have the discovery experience – virtual worlds Amadeus, Hex, The Hex, Boating Demo, Viridian Island etc.

d) Be Interactive – Virtual Worlds Mountain Climbing, Interactive Land, Huge Mountain etc.

e) Share Social Worlds – Oasis Virtual Worlds, Virtual Conference Platform, Something Theater, Pet Shop etc.

[5]Also known as *Neos VR*, to emphasize its relationship with virtual reality.

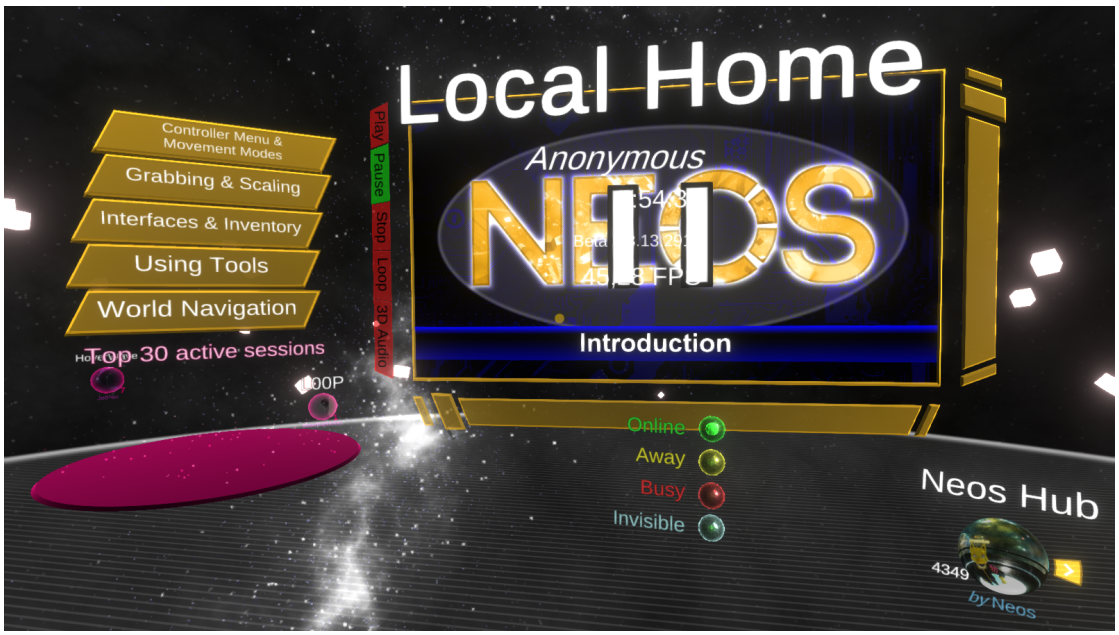


Figure 1.12: Enter world of metaverse Neos.

f) Stay in historic places – virtual worlds The Temple of Artemis, Evening Valley etc.

g) Watch movies, documentaries, concerts, sports, etc.

h) Stream virtual scene content to Twitch platforms, stream TV etc.

i) Perform virtual tourism, use teleportation.

j) Implement engineering and design.

k) Show architectural visualizations (archvis, archviz) – virtual worlds Cube 32, Metatron, Tree etc.

l) Educate and train – educational applications and virtual worlds Blood Stream, Dinosaurs!, Human Anatomy, Carnivores, Virtual Reality Lectures etc. The latter includes worlds VR1 through VR11, which translate most of the content of this book (Chapters 2–4) directly into virtual environment. What you meet can be seen, experienced, tested and developed in these worlds.

m) Operate in v-commerce – virtual reality in trading from car dealers, through retail, virtual shop windows, rehearsals, emotional brand perceptions etc.

n) Participate in mental health, medical applications, slow time therapies etc.

o) Experience Fantasy Worlds – Virtual Worlds Thats Bananas, Cube 32, Swamp Thing etc.

p) Develop a social community of avatars – organize joint events, hold business meetings, help create worlds etc.

q) Work – be an employee, produce, provide services etc.

r) Perform banking services – realize personal ATMs, payments, virtual currencies etc.

s) Investigate – the laws of virtual space, the behavior of avatars, eye movements, etc.

t) Display the depth of the Human Mind – producing personal virtual worlds.

u) Provide digital immortality.

v) Lecture – to hold university lectures, conference appearances, scientific competitions, poster sections etc.

w) Experiment with light, interactive camera, materials, textures, particle systems, 3D models, 3D environments etc.

x) Find use for so many other applications.

y) Consume eroticism, pornography.

1.4 On Virtual Reality and Real World Relationship

The virtualization of the future gives rise to the idea that various human activities will be transferred and performed in virtual environments, and this will become a common part of human lives, just as virtual moneys are used today, paying by card or screen scan where it is not paid by real moneys but by something virtual.

The idea that there may be other worlds than the real one is not new, and has emerged in antiquity, for example in connection with the reincarnation idea. The whole Pythagorean school in ancient Greece was convinced of the transmigration of souls between humans and animals. The consequence was, for example, the strict vegetarianism of school members in order to avoid a situation where someone could enjoy their reincarnated ancestor.

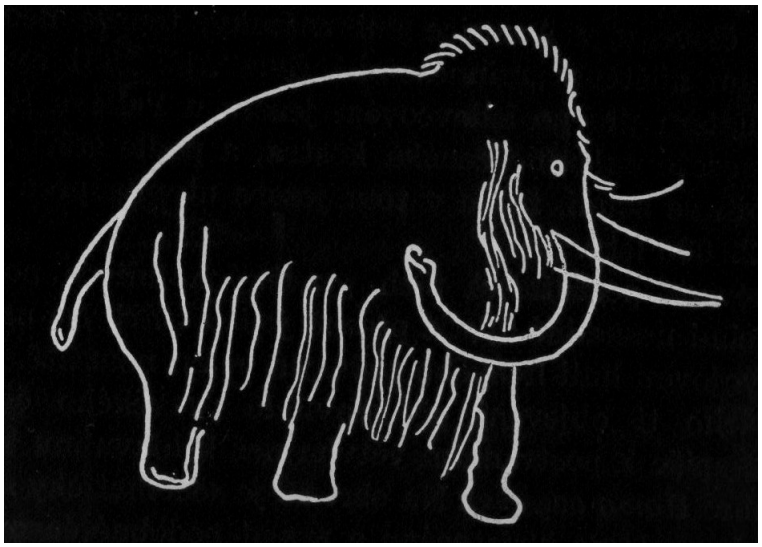


Figure 1.13: Mammoth depiction based on cave painting.

Yet at times around 40 thousand years BC, cave paintings suggest that no other world, or a world beyond our world, was likely to consider[6]. When you study cave paintings from that time (see, for example, the image of the mammoth in Fig. 1.13), there is nothing more than images of real life or images of altered states of consciousness.

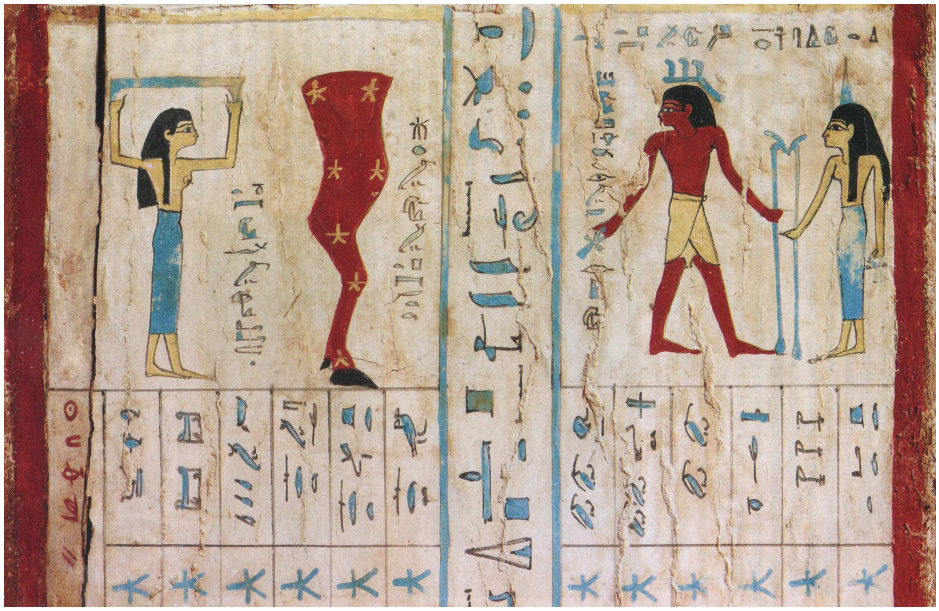


Figure 1.14: A guide to communicating with the world beyond found in Pharaoh's tomb.

A few tens of thousands of years later, in ancient Egypt, the members of the society at that time were clearer, at least the highest members. The pyramids identify an attempt to connect Pharaoh with the world beyond, the world he would enter after he left. This is evidenced by the decoration of tombs hidden deep in the pyramids. An example of a tutorial on communicating deceased Pharaoh with the world beyond dating back to about 2 thousand years BC is shown in 1.14.

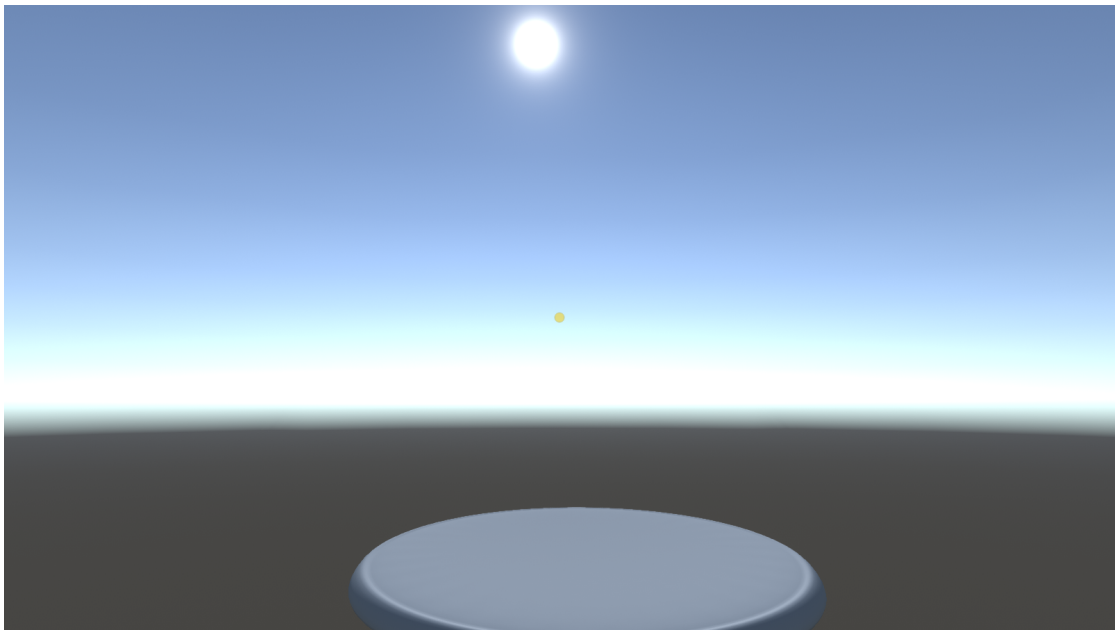


Figure 1.15: Empty virtual world.

For today's virtual reality, there is nothing easier than to open another virtual world that has

[6]Other than altered states of consciousness.

its sky, its land and the sun. You find yourselves in a blank copy of our real space. You stand on a flat surface similar to the lunar one, with a blue sky and sun shine above you (see empty world shown in Fig. 1.15). Empty worlds with earth, sky and sun can be infinitely opened in a virtual environment, filled with a content and communicated with everyone. They may contain copies of parts of the real world or purely fantasy ideas, or something in between, according to the idea of the virtual continuum in 1.7. This new technology therefore has the parameters to benefit human society by decentralizing real space.

When you create and fulfill content of new virtual worlds, where there is earth, sun, sky you are in a way in the position of a Creator as in the figure 1.16. You see the Creator hovering over the Earth, the Sun in one hand, and the Moon in the other, placing them to the sky[7]. Let us therefore believe that virtual worlds will improve life at least in the same way that temple paintings have led to the completion of spaces that overwhelm.



Figure 1.16: The Act of Creation. Graphics. Source: J. Rosendorfský

[7]This graphics is from year 1675. It was made by C. Fantetti sometime a hundred years after Raffael's original. The graphics is located in the Kladruby Monastery as part of a comprehensive cycle of biblical scenes in paintings. It presents a gift of Queen Christine I Swedish to the monastery.

Chapter 2

METAVERSE NEOS



Figure 2.1: Neos metaverse in an abstract concept. Source: unknown author

The Chapter introduces tools and ways of the Neos metaverse for building virtual worlds and creating their content:

- Download and installation.
- Open and fill virtual worlds.
- Use tooltips and brushes for drawing, modeling shapes and providing color palettes.
- Positioning and transformation of objects, terrains.
- Import and saving of 3D virtual scene objects.
- Design and apply particle systems.
- Work with scene inspector and add components.
- Provide scene lighting, photography and streaming.
- Produce materials, textures and shaders.

Fig. 2.1 illustrates the fact that building of virtual worlds requires a lot of components that come together in the metaverse. One of the most important feature of the virtuality is a creative power.

2.1 Download and Install

Before you install Neos, we'll make sure you have an actually updated Windows 10 operating system, as well as the latest graphic card updates, Bluetooth, and other support[1]. The process at the end of which is functional Neos includes four important components:

1. Download and install Windows Mixed Reality (WMR). If you wire glasses to your computer, Windows Mixed Reality will start automatically. This will be the case if you have a WMR headset (for example, Acer WMR, Dell Visor, HP WMR, HP Reverb, Lenovo Explorer, Samsung Odyssey+ etc.). In case of a headset of another category, such as Oculus Rift (S), HTC Vive (Cosmos), Valve Index platform STEAM etc., you install an operating system virtual extension directly from the manufacturer's website, ie `http://www.oculus.com/setup/`, `https://www.vive.com/us/setup/`, STEAM etc.

   The first step that the program will graphically guide you is the so–called controllers *pairing*. The latter indicates their assignment to the computer. Pairing is performed only once at the beginning. Controllers communicate with your computer via Bluetooth[2]. The pairing button is usually located under the battery cover, and the pairing button must be held for a while until the controller LEDs start to flash. Correct connection of glasses and controllers is indicated on the computer screen by the specific window shown in Fig. 2.2.

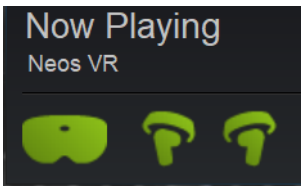


Figure 2.2: Functional glasses and both controllers.

   Next step that will guide you through the graphical environment is to define the area where you will move with glasses. First, it is usually necessary to place the headset in the middle of this area and after confirmation you trace with glasses a border of this area. You start from computer location and go with glasses around the perimeter of the area and close the perimeter at the computer location again. If you're done, you can put on glasses. Windows Mixed Reality offers a simple, natural, pleasant home–landscape world where several applications such as web, video, gallery etc. interactively run.

2. Open STEAM at `https://steampowered.com`. Register, log in and install Steam. Registration, installation of Steam is similar to Neos free of charge. Once installed, look for the SteamVR, the virtual reality tool in STEAM, select it and install it as well.

3. Find and select Neos VR metaverse in STEAM library. Now, when you run Steam[3], Log in, and the STEAM window opens, as shown in Fig. 2.3. On the left you could

[1] Virtual glasses, virtual environment controllers, and the computer form a comprehensive communication and display system that uses a large number of different technologies. As a result of this complexity, a mood of the entire system may be occasional. Therefore, some patience is appropriate both when installing glasses and in later works. For example, you will find that a program has stopped working, failed to connect properly, or the controllers have stopped working, the program will not start as expected, etc. Close patiently all applications, restart the computer (even several times), and gradually start the virtual device.

[2] Controllers can also be added independently as Bluetooth devices in Windows settings.

[3] It is a good idea to restart your computer first.

see the Neos VR software. Then select and start Neos VR. After starting it, turn on the controllers and put on the glasses. You are in the initial world of Neos as shown in Fig. 1.12 while the window in Fig. 2.2 indicates properly switched on and connected controllers.

4. Read the Neos `http://wiki.Neosvr.com` metaverse Wiki – Frequently Asked Questions (FAQ).

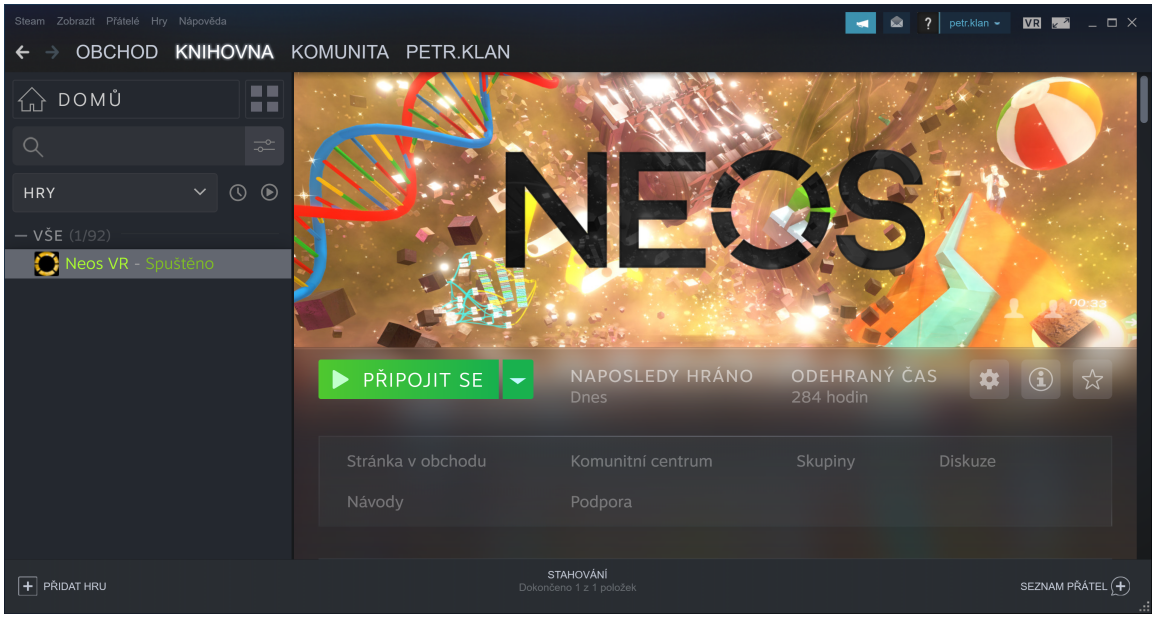


Figure 2.3: Steam window before running the Neos metaverse.

The first hallmark of a proper headset operation is that in the introductory virtual world you see your hands with the controllers, and from each controller a colored light beam focuses on forward. When you move your hands, the virtual hands move together with the light beams.

There are a few additional steps to set up virtual reality for wireless glasses. It is helpful if you have a computer with a wired headset at the same time. This is not necessary, but this will speed up the setup process considerably.

2.2 Introductory World

The immersive experience of virtual reality requires that virtual scenes well follows reality. Probably the first thing to expect will be that the virtual world objects behave like in real world.

The first attribute of such a behavior is the ability to snap or grab an object. If you are absorbed by the introductory world of Fig. 1.12, then you could focus the beam on a scene object and activate the side button with your hand. This function will grab the object and it is possible to move it to another position. Turn off the side button to release the object.

Another attribute is a movement. Virtual space offers more movement options, including

walk, fly and teleport. It seems to be reasonable to choose the way of movement at the beginning. Convenient and quite versatile movement in virtual space ensures flying. To set the way of movement, press thumb small knob of the controller menu and select the motion icon. In doing so, select from items[4] shown in Fig. 2.4:

- Undo: if you turn on Undo, then press the index finger move you a scene backward.
- Redo: if you turn on Redo, then press the index finger move you a scene forward.
- Locomotion: if you turn on the type of movement, then press the index finger to switch between Fly, Walk/Run and Teleport.
- Reset Scale: if you turn on Reset, then press the index finger reset you the scale.
- Duplicate: if you hold the object, turn on Duplicate and press the index finger to confirm, then you make a copy of the object you are holding.
- Destroy: if you hold the object, turn on Destroy and press the index finger to confirm, then you delete the object you are holding.

Press the menu knob repeatedly or give up the controller from to hide the menu. The selection from the controller menu can be described schematically according to Tab. 1.1 as a sequence: thumb → choice → index finger → thumb or giving up.

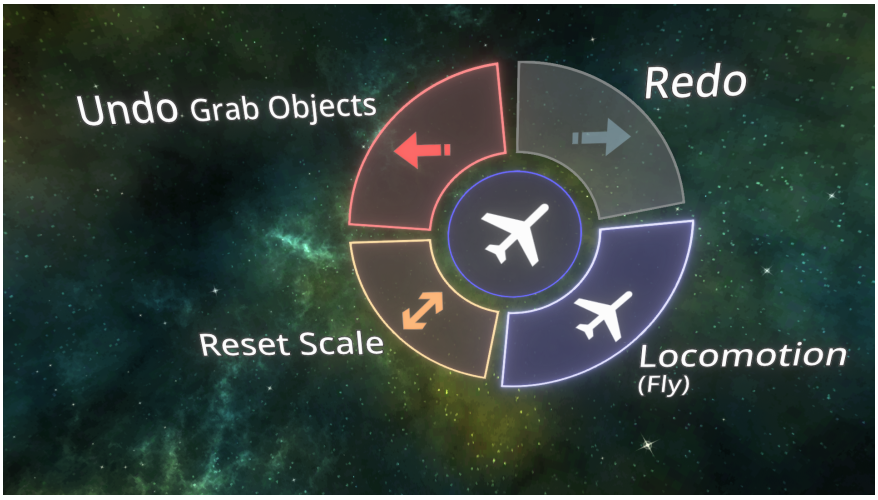


Figure 2.4: Controller menu.

Looking at the introductory virtual world, you could observe a Neos dashboard (or briefly dash) as shown in Fig. 2.5. It represents a primary panel menu that has the same form in all worlds and where you can find everything to create, search and switch worlds and everything to do with their content, including saving and linking them to the external environment. Dash presents navigation as follows:

- New World creates a new, content–free world.
- Save saves actual world.

[4]Note that items Duplicate and Destroy are based on the content, you have them in the controller menu, if you hold an object in scene, for exmple.

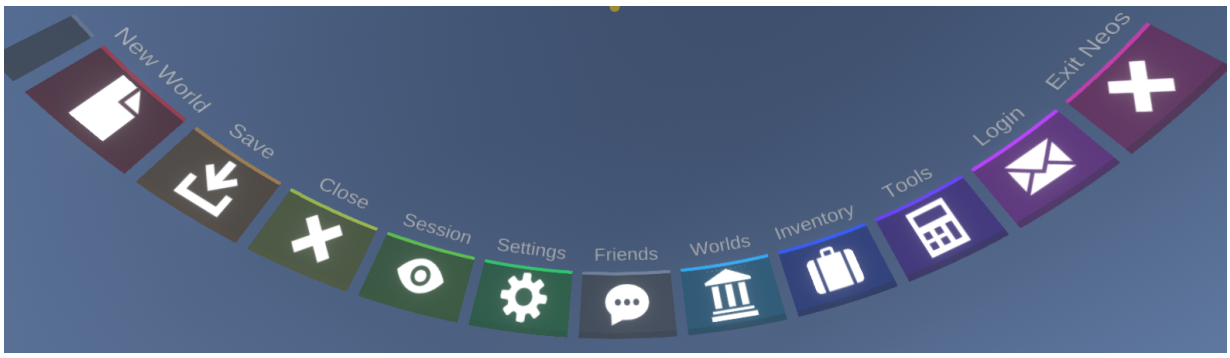


Figure 2.5: The dashboard or dash.

- Close Leave leaves (closes) actual world and moves to parent world level.
- Session sets world parameters, access rights and users.
- Settings adjusts technical parameters.
- Friends communicates with friends in the form of short text messages. If you open Friends, you will receive a window similar to the one in Fig. 2.6 where you can read messages from Neos headquarters, invite friends to discuss, to the worlds, write and read text messages, send objects, worlds etc.



Figure 2.6: Communication with friends.

- Worlds gives list of user's worlds, Content Hub, and public world browser.
- Inventory offers container of tools, models, and a place to save worlds and objects.
- Tools presents tools like file browser, avatar initialization and interactive streaming camera.
- Login opens form to login – username and password.
- Exit Neos closes Neos. You can either save or discard your changes.

In all cases, just focus the controller beam on the dashboard element and click the index finger.

2.3 Virtual Cathedral Hub

A Virtual Cathedral or Content Hub is a multipurpose site which has

1. social character – meetings of users, possibility to play board games,
2. informative character – browsing and searching worlds.

It is also the place where the currently active unfinished worlds are displayed.



Figure 2.7: World browser.

The world browser is shown in Fig. 2.7. On the left side you can meet world categorizations (social, educational, gaming, etc.) and in the white box at the top right you can search worlds by key words. When you focus the beam on the white box and click your index finger, a keyboard appears where you can type an alphanumeric expression. Focus on the letter by the controller laser beam and press the index finger to print the letter.

It is possible to reach cathedral either from the introductory world by focusing the controller beam on the orb Content Hub in Fig. 1.12 and double–clicking the index finger will load the cathedral world. Another option is to go through the Worlds and activate Content Hub. If you require to browse the worlds, you can launch the browser via Worlds and activate Browse Worlds with one click of the index finger. The world browser from the Fig. 2.7 will immediately open.

The cathedral itself is admirable and was built on the basis of cooperation of members of the Neos metaverse community. During the first visit, it is recommended to explore it with a survey of all corners and breathtaking views. One such is captured in Fig. 2.8.

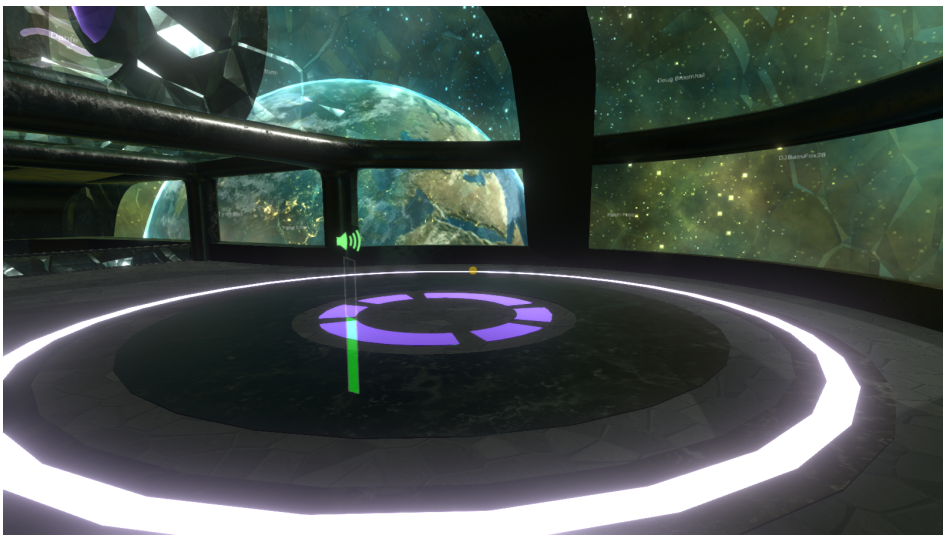


Figure 2.8: View from the cathedral.

2.4 Empty World

Before to start filling of a new world, draw attention to the important immersion rule that the virtual content should be consistent with the real world and keep the definite aspects of it. Even if it is a virtual world with purely abstract fantasy content, it should not suppress the ability of a user to connect with reality. Inconsistencies in the virtual world lead to a disruption in the immersive potential of virtual reality. Generally speaking, although virtual worlds may not represent a mirror of the real world, they should, however, speak the same language.

To create an empty world, use New World from the dashboard menu. After focusing the beam on and clicking with the index finger, a form shown in Fig. 2.9 will be opened. If you want to create world from Fig. 1.15, check (focus the controller beam on and click with your index finger) Basic Empty. In the World name field, name the world and select the access rights to this world (typically Anyone). Focusing on the Start Session and clicking the index finger, you find yourselves in an empty world on a gray earth with the sun and sky above your head. You will have a complete dash available.

The first thing you can do is to save the world in inventory. Open the dashboard inventory by focusing the beam on Inventory and click with your index finger. This will open the inventory folder. Now you can save the new world. Use Save from the dashboard menu, focus the beam on and click the index finger to expand the menu and focus on and click the Save As item. Orb of the saved world will then appear in the inventory. When creating a new content, it is wise to save the world continuously by reactivating Save.

The content of the world consists of *objects* and their behavior. In the empty world of Fig. 1.15, there are objects, such as sun which illuminates the world and a static earth. Other objects can be either created or taken from the Neos metaverse inventory, or downloaded from somewhere and imported into the world. A system of objects and their behavior forms *scene*. Worlds typically contain multiple scenes. For example, in one part of the world, it may be cloudy, windy, rainy, and in another part, the sun shines and tempts world visitors explore a jungle.



Figure 2.9: A form of the empty world.

Scene objects have a hierarchy. For example, one object can be embedded in another (parenting). You can pour tea into a cup, make a cave in the mountain, make an island in the sea, have birds on a tree, write a mathematical formula on a T–shirt, etc. The relationship of such objects is usually expressed by a parent–child pair. The object at the root is called the *parent object*, and any object below in hierarchy his children. As you will see later, the above object hierarchy plays an important role in working with objects and creating their behavior.

So let us embark on the adventure of creating worlds and filling them with objects. It is because, for the first time in history, there is an opportunity to create worlds with space that models real ones. People once enslaved letters to create words to model stories. There were a number of critical voices at that time, afraid that they would write all lies. Yet the words that developed our culture are the winner. Virtual reality enslaved millions of transistors and elementary number operations to create the real space model. Even now there are voices that the virtual reality will be primarily important to satisfy the lowest instincts of man. Yet virtual reality will be a winner in the development of human culture towards fantasy and approach to the depth of the human mind.

2.5 First Steps in New Worlds

You can immediately fill the empty world you created with a content. The first tooltips you probably use will be brushes and tools for the basic shapes of objects such as cubes, spheres, cylinders, cones etc.

As in the real world, you could expect virtual reality tooltips to do what they're meant to do. To access the tooltips, open Inventory and double–click Essential Tools with your index finger. The tooltips inventory opens as shown in Fig. 2.10.

First select a tooltip by the beam. When you use a single click by the index finger, you confirm the tooltip. Its icon is then highlighted and at the top of the inventory window, you could see a short text message to identify the tooltip. The second, a double click will show



Figure 2.10: Inventory of basic tooltips.

the tooltip separately beside the inventory.

Before using the tool, you have to take the separate tooltip into hand. To do this, focus the beam on the tooltip and press the side button of the controller twice in a row. You notice the change because the tooltip appears in your virtual hand. When you move the hand, you move the tooltip at the same time. Now, the tooltip can be used for the requested purpose.

Tooltips are typically used by pressing the index finger and moving the controller. For example, to use the Cube Builder, press the index finger to create a small cube, and drag the controller while pressing the index finger to size it as needed. Then release the index finger.

If you later find that an object is too small or too large, you can resize it by focusing both controller beams on the object at the same time. When you press the side button on both controllerss and drag the controllers away from each other, you either enlarge or reduce the object.

Objects can be moved as needed in the virtual world. When you focus on the object with a beam and press the side button, you snap the object and move it to another place. In addition, the controller joystick can be used to zoom it in or out. Release the object by opening the side button of the controller. When you need to pick up an object, raise and rotate it, for example, you focus the beam on the object and press the side button. The object can now be raised and rotated because the object will follow the movement of the controller.

If you take a brush into the hand, a press intensity of the index finger affects the strength of the line: a slight pressing realtes to a thin line, while a stronger pressing makes the thicker line[5].

Example 2.1 *Pyramid.* Build a 4×4 pyramid in the base, 3×3 above it, 2×2 above

[5]Note that there is a ruler in the inventory to draw straight lines as well.

it, and a 1 cube on the top. Draw simple mathematical symbols on one wall of the pyramid.

Use the tooltip for basic shapes to create cube of an appropriate size and make 29 copies $(16 + 9 + 4)$ using Duplicate in the controller menu (altogether 30 cubes). Snap each cube and use the joystick to move it to the specified location. You build the base 4×4, the second layer 3×3, the third layer 2×2, and move the last cube to the top. Take a drawing brush and draw a simple mathematical symbol on each cube. It results in a pyramid shown in Fig. 2.11.

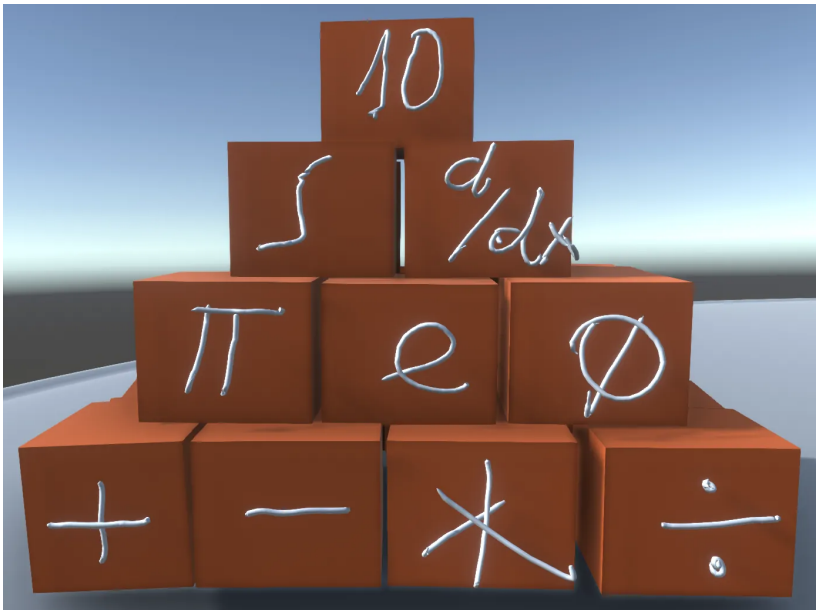


Figure 2.11: Pyramid with mathematical symbols.

2.6 Gallery of Tooltips

A set of powerful purpose–built tooltips is available in the Essential Tools folder:

a) *Basic shapes:* tooltip for basic object shapes such as cube, sphere, cylinder, cone, surface is shown in Fig. 2.12 left. By repeatedly pressing the large knob of secondary action on the controller, the tooltip gradually becomes a source for creating cubes, spheres, cylinders, cones or surfaces. Actual shape of the object is displayed directly by the tooltip.

b) *Luminaires:* tooltip from Fig. 2.12 right becomes a source of spherical lights after taking the tooltip into the hand (double pressing the side button). To create a new light simply press the index finger. The lights light up scenes just like the real lights. Individual lights can be switched off and on (they represent logical variable with values 1 (TRUE) – on or 0 (FALSE) – off. To do this, focus beam on the light and press the index finger.

c) *Particle effects:* tooltip works as a spray gun. For example, when you press the index finger, the tool emits a dense and focused stream of light particles to a great distance,

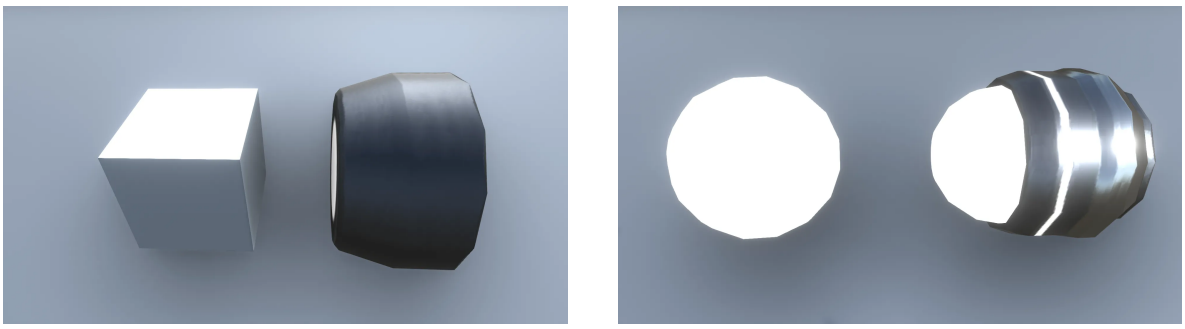


Figure 2.12: Left: tooltip for creating objects of basic shapes. Right: light creation tooltip.

as it is shown in Fig. 2.13 left. Thanks to gravity, the stream naturally follows the ballistic curve just like a narrow, intense jet of water from the spray gun. This creates an impressive light effect.

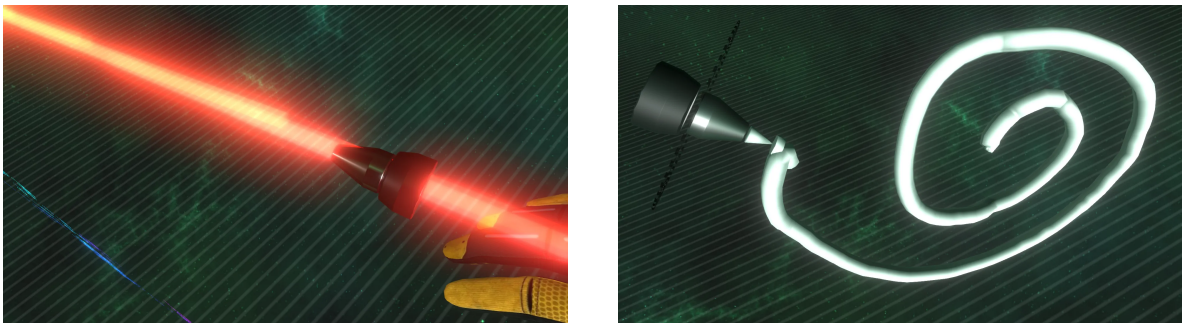


Figure 2.13: Left: light effect tooltip. Right: drawing brush.

d) *Drawing:* grab the drawing brush by pressing the side button twice and by activating the index finger. You start drawing as in case of a spiral in Fig. 2.13 right. The strong or weak line depends on the force of pressing the index finger. If you press the controller menu knob, you could see the Change Color item in the circular menu. If you confirm it by a click of the index finger, you get a possibility to change color by the so called *albedo*. By focusing the beam on, pressing the index finger and pulling, you can rotate the triangle in a circle to change RGB (red/green/blue) color or to move the potentiometers in the basic color intensities. You can also use an alpha potentiometer at the bottom for opacity of the mixed color or its degree of transparency. In this context it is talked about RGBA system (red/green/blue/alpha), where all RGBA values are in the range of 0 – 1. For example, $(1, 1, 1, 0.1)$ is a low–white, transparent, while $(1, 1, 1, 0.9)$ represents an intensive, opaque white. If you confirm the color at the bottom of the form, the brush lines will be as colored as the glowing white color in Fig. 2.13 right.

e) *Gluing:* glue tooltip is a frequently applied tool. It is used when you have multiple objects and need to glue them into a single object. Gluing ensures that a single object is created from several objects, for example for a single moving. The gluing tool is shown in Fig. 2.14 left. Grab the tool by double pressing the side button of the controller and place it where you want to stick objects together. When you press the index finger, a small white ball appears on the tooltip. It is increased by pulling the controller to cover the glued place. Release the index finger, the white ball remains in the glued place. Wait for a moment and the white transparent ball evaporates spontaneously. Once disappeared, the parts are firmly glued together.

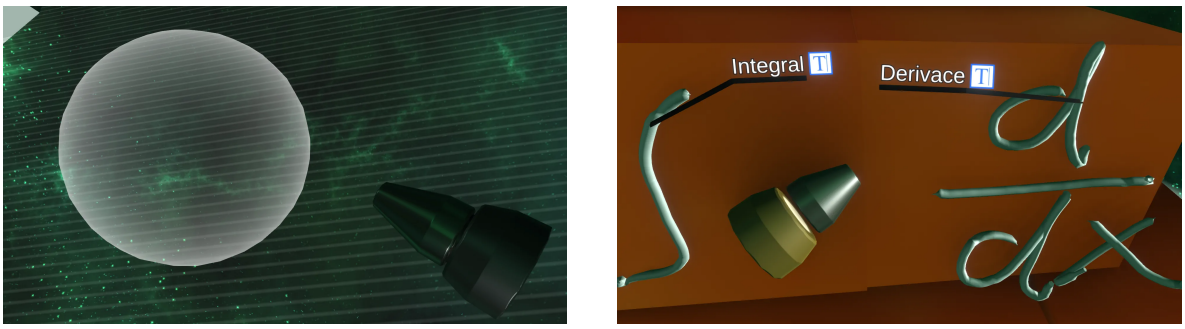


Figure 2.14: Left: gluing tool. Right: Labeller.

f) *Labeling:* another tooltip used is a marker or labeller. After grabbing it, focusing the beam on and clicking the index finger, a label will appear at the marked location, into which a short text can be written. Focusing the controller beam on the label field and clicking the index finger, the keyboard appears. Select a letter with the beam and click the index finger to write the letter in the label area. It models situation when you press a key. For easier orientation in this typing mode, the characters appear in the upper bar of the keyboard in parallel. Pressing Enter closes the keyboard visualisation. Two text labels made in the above manner are shown in Fig. 2.14 right as the labels of some mathematical symbols shown in Fig. 2.11.

g) *Distance Measurement:* tooltip in Fig. 2.15 left is used to measure distances between objects. After grabbing the tool by pressing the side button twice, you focus the beam first on the first point where you want to measure the distance. A marked point appears by the clicking the index finger in a given place. Move the beam to the opposite point of the distance and click with the index finger. A distance dimension is automatically measured between these marked locations.

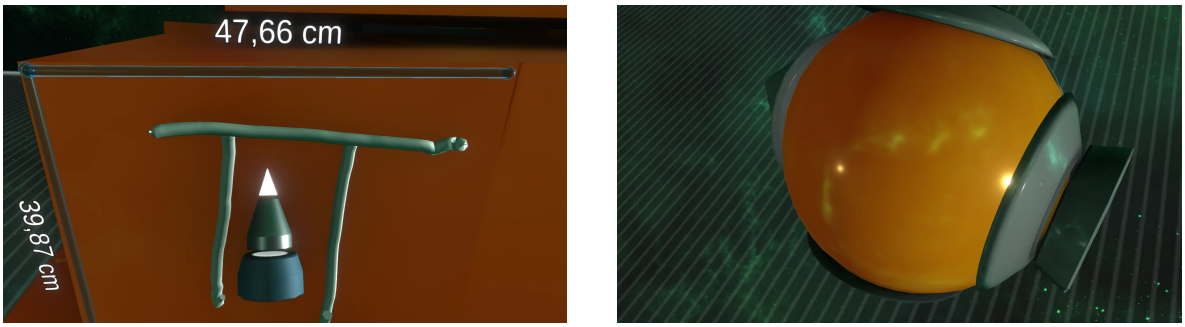


Figure 2.15: Left: distance measurement tooltip. Right: Material tooltip.

h) *Materials:* in a virtual environment, objects have surfaces, similar to the real environment. Each surface can be filled with some material, for example, simply color the surface with painting. To do this, use the material tool in the form of Fig. 2.15 right. The material tool itself is characterized by an empty circular stack, a free space where you put the material you want to use. Materials are represented by small balls whose surface illustrates the material appearance. Working with a material tooltip begins when you grab the tooltip by your hand typically by pressing the side button of the controller twice. Now you need to charge the empty stack with material. By pressing the side button of the controller, in which you do not hold the material tooltip, grab the material ball and insert it into stack. After proper insertion, the material ball merges

with the toolltip so that when you move the tool, it holds firmly in the stack and does not fall out. With this charge, you focus the beam of the controller on a surface. Clicking the index finger fills the surface with material from the stack. Material in Fig. 2.15 right is the material that was used to color the surfaces of the cubes of the pyramid in Fig. 2.11.

An extensive set of materials is offered by the inventory folder called Materials. To get there, open the inventory and double–click on Neos Essentials. Or use inventory navigation arrows. When you get to the Materials folder, you recognize the variety of materials by name. If you select a materials, then double–click on it causes that the material ball appears next to the inventory. Then use the ball to charge the material tool. Finally, most materials are given by the so called CC0 textures, which are freely available at `https://cc0textures.com`. They mostly are high–quality materials, so it may take seconds to transfer them. Therefore, after double–clicking on the selected material, wait patiently for the material ball to appear.

i) *Decoration:* in Fig. 2.16 left, there are two decorative tooltip brushes: the first is a source for clusters of small cubes and the second is for clusters of sheets. The first is used for decoration of the environment, the second for decoration of natural objects such as shrubs or trees. Both tooltips are grabbed by pressing the side button. Each time the index finger is activated, the cluster will come out of the tool. Fig. 2.16 right shows a tooltip that models free convex shapes. Grab this tooltip and press the index finger. You create a fancy convex shapes by the hand pulling. When you release the index finger, you can work with the convex solid body as with a current object. In Fig. 2.16, two independently created solids are shown. Therefore, the tool is useful as a source of stones, towers, swords etc.

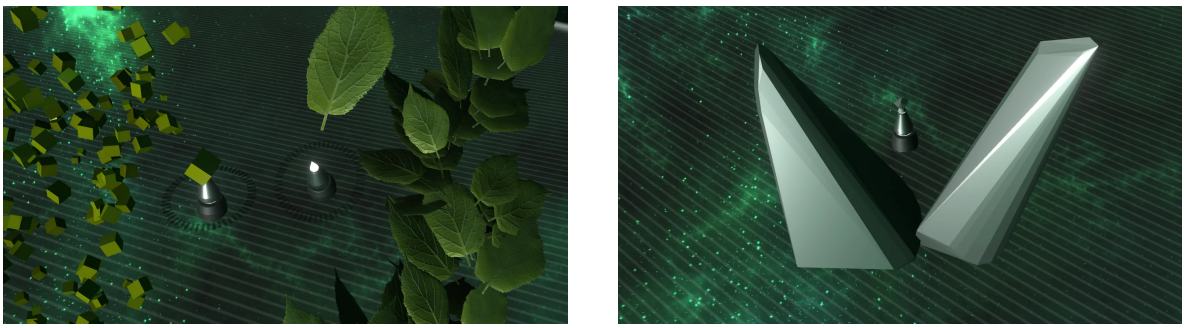


Figure 2.16: Left: tooltipss for clusters of cubes and sheets. Right: convex shape tooltip.

To conclude, notice that the tooltip in your hands can be released by pressing the side button twice, ie in the same way when grabbing the tooltip before using it.

2.7 Best practices of building virtual worlds

In the previous chapter, two essential attributes of virtual worlds were mentioned: the ability to immense and the presence. Virtual scenes should not be built in such a way as to substantially disrupt these two attributes. Give therefore a collection of rules that are wise to follow when you buld virtual worlds. On the other hand note that virtual reality is new medium

where much experimentation and new approaches are constantly emerging. Therefore, it is desirable to follow recommendations below. Note that they will probably be outdated by ideas that are not yet known.

a) Minimal vection. The latter creates an illusion of movement yourself caused by a move of large scene objects. It can lead to a nausea. This type of illusion arises when you are sitting in a car and watch the movement of a large vehicle, such as a truck or a train, from the window and feel that you are moving in the opposite direction instead. Therefore, a caution must be exercised in any movement of objects that fill much of the virtual scene. In this sense, for example, a smooth rotation of the entire scene relative to the static user is really problematic. In this situation, snap the user to rotate as well since it relieves the illusion.

b) Suppression of stairs. Move up the stairs can lead to a feeling of vertical movement, where at the end of it you may not have a pleasant feeling, especially when you reach higher heights faster.

c) Using dim light and colors. Use muted colors and cooler shades than you would normally use. Too bright lights can lead to a nausea, and sharp contrasts can contribute to creating a virtual world moving towards the user.

d) Maintaining accurate scales. Virtual world objects should be as small or as large as real–world objects. Excessive differences can lead to a nausea. Therefore, accurate scaling of the virtual world objects is important, and the length measuring tooltip can be used if needed. Note that a basic unit of Neos metaverse length is represented by 1 m.

e) Reducing eye strain. Excessive eye strain can lead to headaches. One cause of eye strain is blinking. For example, high latency causes blinking. Therefore, you must keep it at a low level, as mentioned in the previous Chapter. The eyes also focus on objects in virtual space. Therefore, it is essential to keep the minimum important object distances from the user's eyes at ideal value of 1 m. At the same time, it is important not to expose users to a long–term view of distant objects of size less than 0.5 m.

f) Trading off virtual scene content and its perception. In a virtual environment, it is easy to conjecture flights, falls or militancies. Virtual reality is therefore much more capable of raising fears or an intense sense of fear than a flat screen. This is because you are immensed in the virtual environment, unlike the flat screen, where peripheral vision tells you that what you are watching may not be real. Feelings of presence in virtual reality can therefore lead to stronger reactions. In virtual reality, unlike film, you could be fully present. This instinctively instills in you feelings of a personal space that you can use for some great purposes. This space seems to be real, because senses of behind you or before you mean exactly the same as in reality.

g) Breaking in use. Virtual reality puts more strain on the body, eyes and mind than other media. You have a headset on the head, often standing and moving physically. It is therefore important to have a freedom to interrupt the virtual experience at any time and to return to it if wished. This is related to downloading of worlds, which should be as short as possible. Therefore, it is important to optimize the size of the worlds with respect to the speed of downloading.

h) Building of a minimum viable world. Virtual building is usually preceded by an identification of target users and a well–answered question why just virtual reality and not other media. In doing so, it is recommended to build a *Minimum Viable Virtual World* (MVVW). It is an initial world that is of a great importance for further development. It contains only the most important things to fulfill its purpose and little or nothing

else. MVVW thus represents a backbone of the content, specifying its priorities and appreciation. User's response to MVVW published in a common virtual hub for instance is therefore one of the most important.

i) Decomposing world content into parts. It is technique of early detection of problems. You will divide the world you plan to build into smaller parts (breakdown). This will give you a list of items that you need to create in the virtual environment. You will make sure which parts you have time and resources for, what you must not forget, what can be problematic, what items are important for MVVW, what can be omitted at this stage, etc.

j) Testing early and often. Virtual reality presents new medium which reacts in different ways. Therefore, it is appropriate to test the worlds at early stages of their emergence under the widest possible audience. Testing is useful both for users who have some experience with virtual reality and for users who will be the first experience. In an immensive virtual world, the first experience can be realised to be close to an alien experience.

k) Believing shadows. Good work with shadows around objects on which falls light helps to reduce inconsistencies with real worlds. Shadows should be believable to our senses. A typical example of noticing that something is wrong with the shadows is represented by the Enigma of the Hour painting. G. de Chirico's image (see 2.17) shows a clock showing 14:54, but the shadows around the clock point to a sunset rather than to a nearby afternoon.

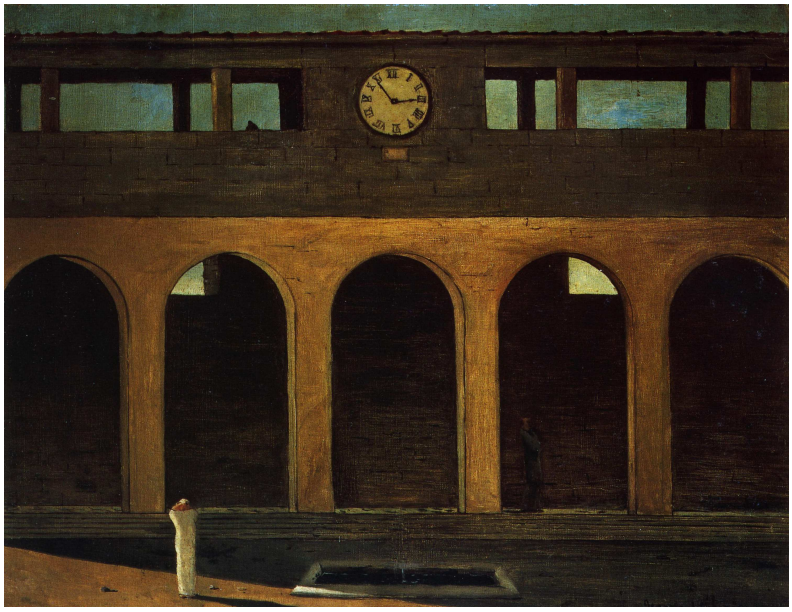


Figure 2.17: The Enigma of the Hour. Source: `https://en.wikipedia.org`

Example 2.2 *A virtual portrait of Mars environment.* Parts of the portrait will be: Mars environment in a specific area based on available NASA data, photos and videos; basic facts, craters, core of Mars, models of probes that landed on Mars, architectural visualizations of possible settlements, etc.

Initial MVVW may contain 3D environment on Mars with views to the horizon at a correct scale as in Fig. 2.18. In doing so, it will be possible to move on the surface of Mars under

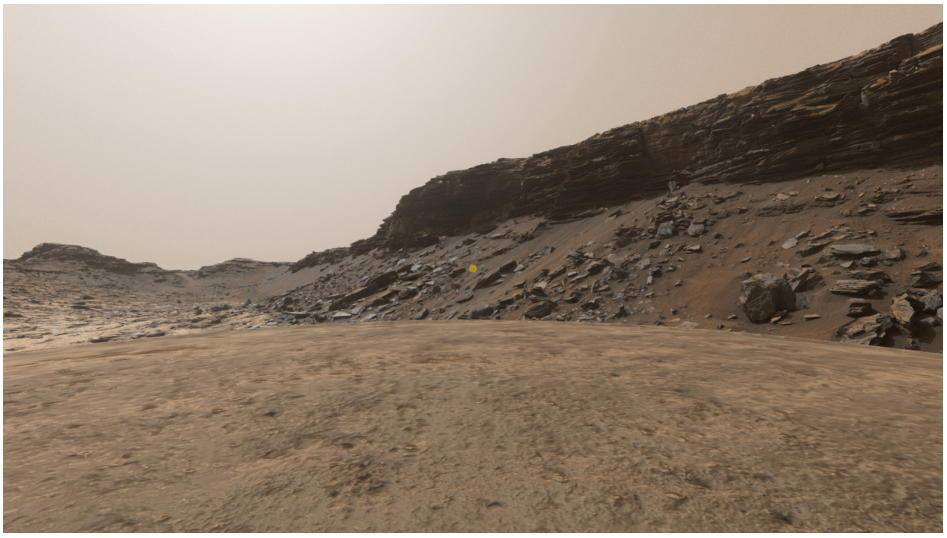


Figure 2.18: Virtual reality of Mars environment.

appropriate gravity. This gives the visitor a concrete idea of how the Mars environment and movement in it look. It presents a minimal viable virtual world that can be gradually fulfilled.

2.8 Interior and exterior adjustments

When you come to an empty virtual interior space as in Fig. 2.19 left, you could realize:

1. Material treatment of walls. Grab the material tooltip (see Fig. 2.15 right) and choose materials in the Materials folder of the inventory, such as a variant of natural bricks. Double–click to confirm the material selection and wait for the material ball to appear next to the inventory. Grab it and use as a charge to the material tooltip. Focus it on the wall and press the index finger. Now, the charged material will be transferred to the wall. If you are ready with wall decorations, it is time to go to the next item.

2. Installation of furniture. Select the Neos Essentials folder from the inventory and proceed to the 3D Models. Here, you find the Furniture subfolder. When you open it, you get a basic menu of furniture such as cabinets, tables, chairs, seats, stools, etc. Select an item (single click of the index finger) and choose a piece of furniture by double–clicking the index finger. The piece of furniture will then appear in the actual size next to the inventory. Snap the piece, rotate it correctly, place it on the ground and move to the designated place. This proceeds piece–by–piece until the interior or its part is fully equipped as shown in Fig. 2.19 right. Note that the content of the other folders of inventory could also help with furniture supplements.

Now, you fill the empty exterior as in Fig. 2.20 left with natural content in a similar way as the interior. In the inventory you could find a Nature folder, choose an object (for example a tree) with a single click of the index finger and then activate it with a double click. The object will appear next to the inventory in the actual size. It can be enlarged or reduced by focusing the beams of both hands on it, pressing the side buttons of the controllers and then pulling the controllers apart (zoom in) or toward each other (zoom out). Then move

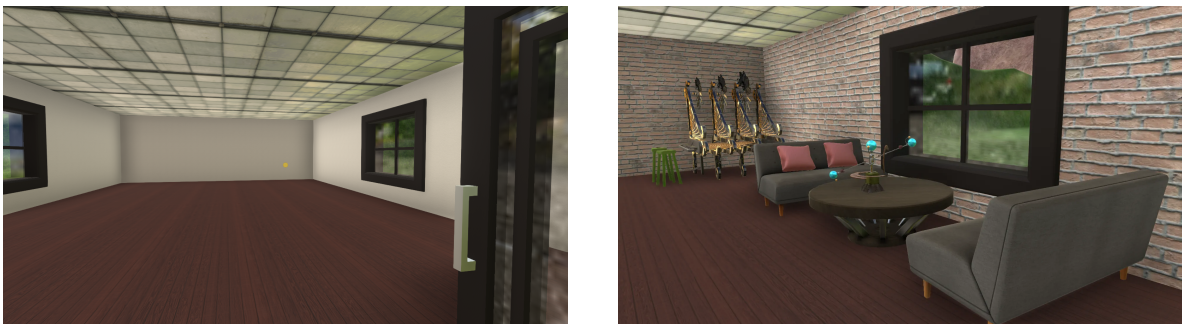


Figure 2.19: Left: empty hall. Right: equipped hall.

the object to the specific location. Objects can be duplicated by grabbing the object and pressing the small knob of the controller with your thumb. Confirm Duplicate (see Fig. 2.4) and press the index finger to double the object. Press the small knob repeatedly or move controller away trom to close the controller menu. In Fig. 2.20 right you could see several trees and a stone circular fireplace with a burning fire.

Fire can be found in the 3D Models folder, the Light subfolder. Stones and many other natural objects can be found in the virtual world of Nature Assets, which can be found in the virtual cathedral hub. When you like a natural object, you add the object to your inventory (snap the object with one controller and press the + right in the top bar of the inventory window with the second controller). This will bring a way the object transfers to your world since choose it in the inventory like other objects.

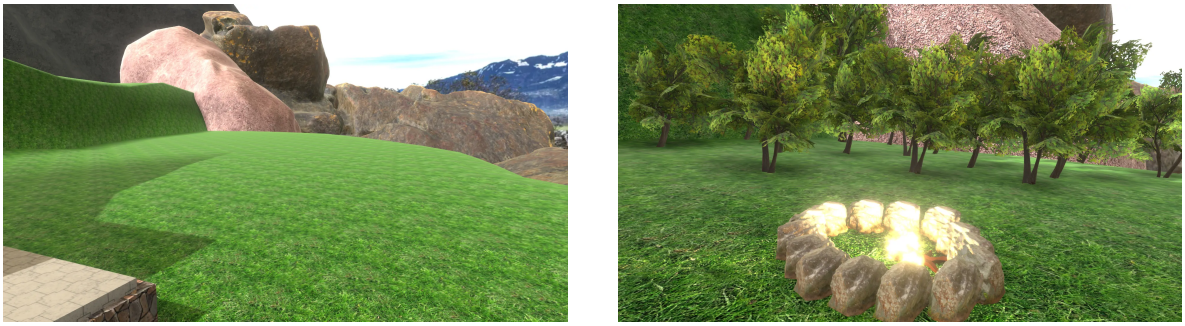


Figure 2.20: Left: empty exterior. Right: natural still life.

2.9 Positioning of objects in scenes

Every work with objects lies in the foundations of virtual reality. Each object that exists in the virtual environment represents a virtual space object. Virtual space is a pure mathematical construct. Each point of an object, all its properties, movements, and interactions are performed using numbers. It may seem that objects exist randomly located in virtual space. However, virtual space has three dimensions similar to te real one, and the objects it contains lie in orthogonal coordinate systems, sometimes called grids.

The best–known spaces are 2D (two–dimensional) and 3D (three–dimensional). Two–dimensional

2D space represents a surface. In 2D space, each motion can be decomposed into two directions: horizontal and vertical (right–left, up–down). For example, JPG images or Tetris game are 2D.

Real space has an extra dimension, depth. It is 3D because every motion can be decomposed into three items: right–left, up–down and forward–backward. For example, a cube is three–dimensional. Metaverse Neos is a 3D machine as well. All the virtual worlds it contains use a model of real space. Therefore, they inherently have three dimensions. Metaverse covers 2D systems, too. This means that 2D elements such as quads can be used in three–dimensional worlds, just as in the real world posters or photographs are used, for example. In addition, screenshots or material textures of the objects also are 2D and can be thought of as a packaging of 3D cans.

Dimensional systems in virtual space are represented by an orthogonal coordinate system. Each virtual space object is located in such a coordinate system and each of them uses lines which are called axes and points which are defined by these coordinates.

An orthogonal coordinate system or *Cartesian coordinate system*[6] in 2D consists of two perpendicular lines called x (horizontal) axis and y (vertical) axis. Both axes are provided with a symmetrical scale, placing 0 at the crossing point. The crossing point is also referred to as the *origin* or starting point. An example of this is the coordinate system shown in Fig. 2.21 left. Similarly, the 3D coordinate system is formed by three mutually perpendicular axes x, y and z intersecting at one point as shown in Fig. 2.21 right. At this point of crossing the origin is placed.

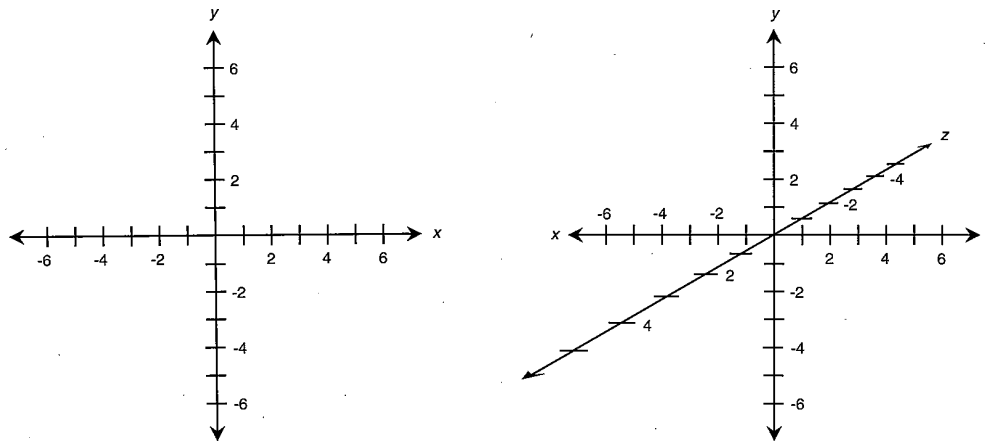


Figure 2.21: Left: 2D coordinate system. Right: 3D coordinate system.

While in 2D coordinate systems you can move in directions of the two axes, ie you draw for example a square, in the 3D systems you can move in three axes, ie you create for example a cube. In a 3D system, the area specified by the x and y axes can be rotated around the z axis.

In 2D coordinate systems, each point can be unambiguously determined by a pair of numbers relative to the origin as shown in Fig. 2.22. Therefore, the origin in coordinate systems is very important. The first number indicates distance of the point on the x axis from the origin, the second number on the y axis from the origin. These numbers are called *coordinates*

[6]Cartesian is called after its discoverer, mathematician and philosopher Ren Descart. More correctly, however, it should be named *Fermat* by Pierre de Fermat, a mathematician who used such a system ten years earlier.

of the point. Origins are determined by pairs $(0, 0)$[7]. The following points are shown in Fig. 2.22:

- $(2, 2)$, that is distance 2 units to the right of the origin on the x axis and distance 2 units up from the origin on the y axis,
- $(-3, 3)$, that is distance 3 units left from the origin on the x axis and distance 3 units up from the origin on the y axis,
- $(2, -2)$, that is distance 2 units to the right of the origin on the x axis and distance 2 units down from the origin on the y axis.

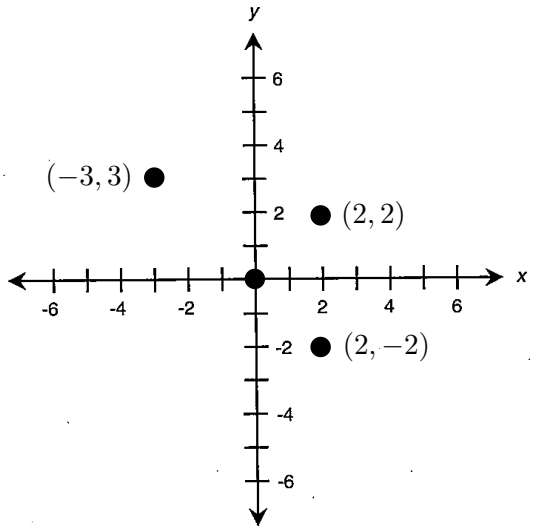


Figure 2.22: Determining points relative to the origin.

The further the point moves away from the origin of the coordinate system, the greater its coordinates are. Analogously, in a 3D coordinate system, each point is determined by three numbers relative to the origin. The first number specifies distance on the x axis, the second on the y axis, and the third on the z axis. For example, origins of the 3D system are determined by the $(0, 0, 0)$ triple, that is distance 0 units right–left from the origin on the x axis, distance 0 units up–down from the origin on the y axis and distance 0 units forward–backward on the z axis (see Fig. 2.23).

It results in that each point P in virtual 3D space is fully determined by its (x_P, y_P, z_P) coordinates in the Cartesian coordinate system as shown in Fig. 2.23.

It would seem that a single global coordinate system is used in virtual worlds. Thus, each point of the scene is determined by 3 coordinates x, y, z at every time and all virtual scene objects share the origin of this coordinate system. However, another coordinate systems form single objects of the virtual scene. They are named local coordinate systems. Each object of the scene is associated with such a local system. The latter is completely separate from local systems of the other objects. It means that each local coordinate system has inherent object axes and an origin that other objects cannot use. Fig. 2.24 illustrates use of a global coordinate system on a square (left) and simultaneous use of both the global and local system (right). Both coordinate systems are needed for example when transforming or parenting objects.

[7]Alternatively, an origin can be determined locally or relatively by non–zero coordinates (x_0, y_0) as well. In this case, distances are determined as the differences $(x - x_0, y - y_0)$.

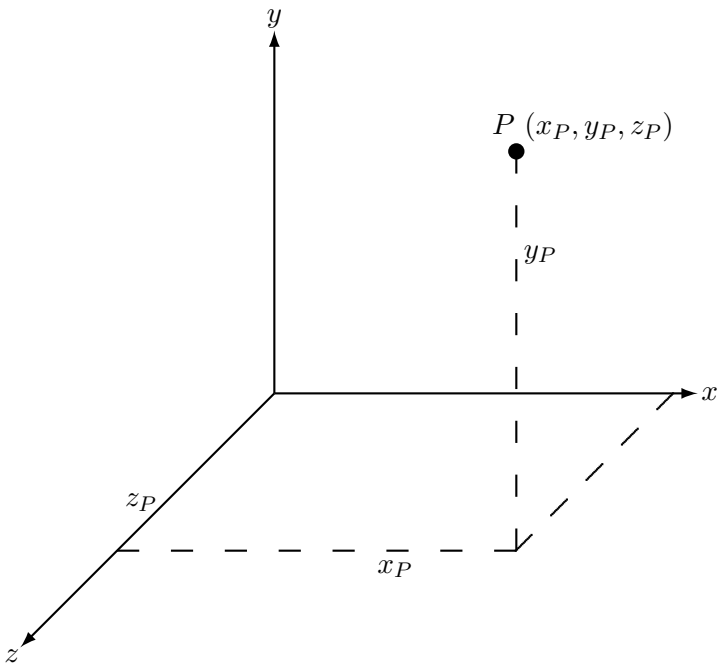


Figure 2.23: Point in the 3D coordinate system.

2.10 Transformation of objects

Every tree, stone, building, flower, chair, lamp, poster etc. in the virtual scene has one thing in common. They determine *objects* as basic elements of virtual scenes, no matter if they are simple or complex. As you could see, the inventory includes a number of built–in objects that can be used to build virtual worlds. A use of coordinate systems allows all virtual scene objects to have

- placement or *position*,
- turning or *rotation*,
- measure or *scale*.

Transformation of an object relates to their change. You transform an object if

- you move the object from one place to another,
- you rotate the object,
- you enlarge or reduce the object in size.

Each virtual scene object can be transformed in such a way.

Moving an object in a 3D coordinate system (translation) from one location to another is the simplest transformation you can apply to an object. Fig. 2.25 shows the translation of the square along the x axis.

Rotation in a 3D coordinate system means rotating around one of the x, y, z axes by a fixed angle. Rotation changes the orientation of the object. If you do not know which axis belongs to centre of rotation, for example the pinwheel of a windmill shown in Fig. 1.3, you can rely on the colors of the axes: x – red, y – green and z – blue. Or you can proceed rotation

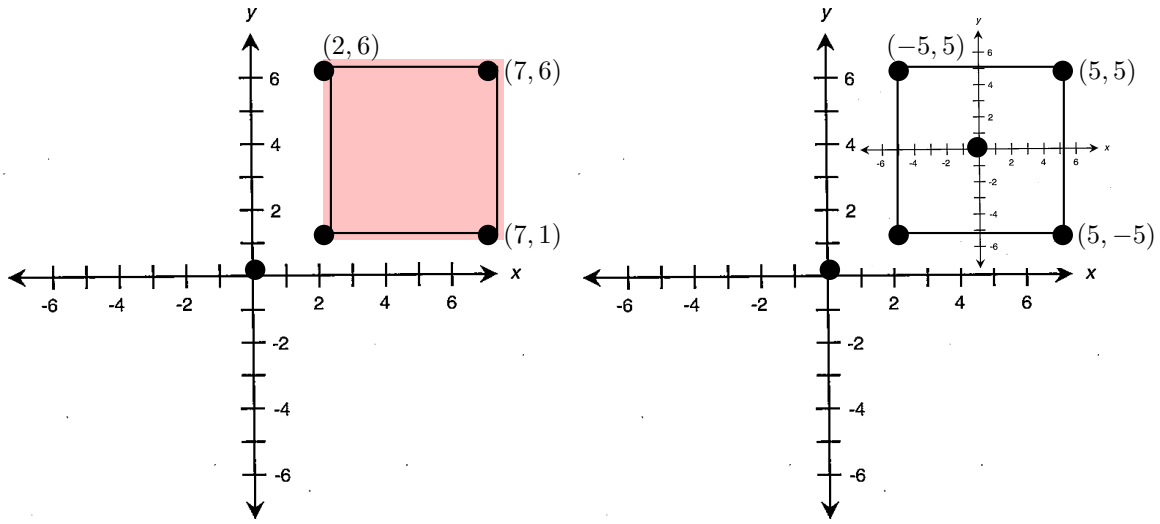


Figure 2.24: Global versus local coordinate system.

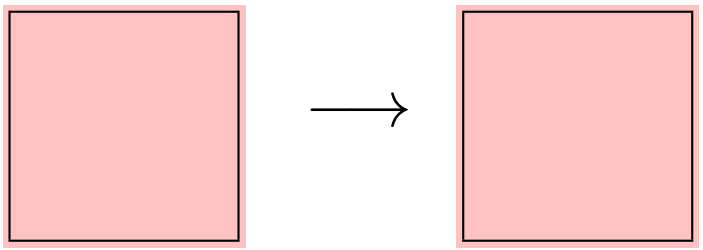


Figure 2.25: Translation of an object along the x axis.

experimentally by a trial and error. Fig. 2.26 illustrates rotation of a square around the z axis by 45°.

Scaling increases or decreases objects in size. You can scale in direction of individual axes x, y, z. Fig. 2.27 shows the size reduction of a square in the x and y axes.

One way to move an object from one position to another is to use symbolics of gizmo. Fig. 2.28 left shows such a gizmo for cube translation. In this case, gizmo refers to a visualized coordinate system. Focus the beam on one of the axes, click the index finger to highlight it, and move the object in the direction of the selected axis. Similar to the translating gizmo, the rotating one can be used for rotation as shown in Fig. 2.28 in the middle. Focus the beam on one of the circles corresponding to the axes of rotation, click the index finger and drag the controller to rotate the object around the axis. Scaling in x, y, z can be done similarly using the scaling gizmo in Fig. 2.28 right. Focus the beam on one of the axes, click your index finger and drag controller on one side or the other to either increase or decrease the object size in following of axis.

Another way to perform object transformations is to use the so–called scene inspector, where you enter data about object transformation directly into the inspector form (see the section of the scene inspector in the next).

Transformations use local coordinate systems of objects. The order of operations is really important. For example, if you first move an object up in the y direction and then rotate by 180°, it will remain at the top. If you swap the operations and first rotate the object by 180° and then want to move up, ie in the positive direction of the y axis, you actually move the

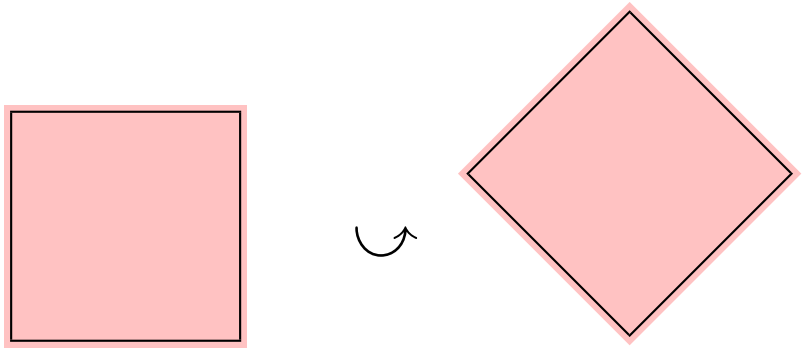


Figure 2.26: Rotation of the square around the z axis.

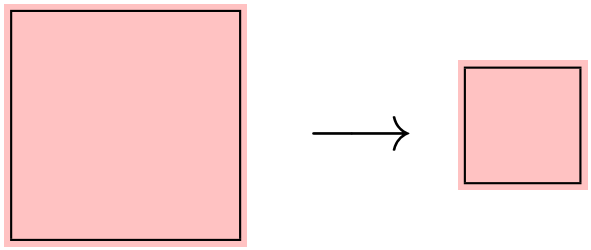


Figure 2.27: Reduction of the square by half size in the x, y axes.

object down. The result indicates that the object remains at the bottom and not at the top, as in the previous case. Mathematically, it would be stated that associative law destroys in this case.

Scaling resizes grid of local coordinate systems. If you increase an object, the coordinate system will increase as well. This change causes the object to grow. The scaling changes are multiplicative. If the scale of the object is 1, it is the natural size of the object. For example, changing the scale to 4 in all axes will enlarge the object four times. If you shift such an object in the x direction by 1 unit, it actually moves by 4 because the local coordinate system has a quadruple grid since $4 \times 1 = 4$. Conversely, if you reduce the scaling to 0.5 and then shift 1 unit, it actually shifts $0.5 \times 1 = 0.5$ unit.

If you insert one object into another object, such as the interior furnishing in the house in Fig. 2.19, the house object becomes the parent object and interior objects by its children. Transformations applied to a parent object – translations, rotations, scaling – are transferred to children's objects. If you move a parent object, its child objects will move with it. However, the position of the children's objects, which is relative to the parent object, will be preserved. If you perform any transformation of an object, you do it not on the object but on the coordinate system of the object. For example, you do not rotate objects, but their coordinate systems. The rotation of the object is only due to the rotation of the coordinate system. Therefore, if a child's local coordinate system depends on the parent's local coordinate system, then any change in the parent's object will be immediately reflected on the child.

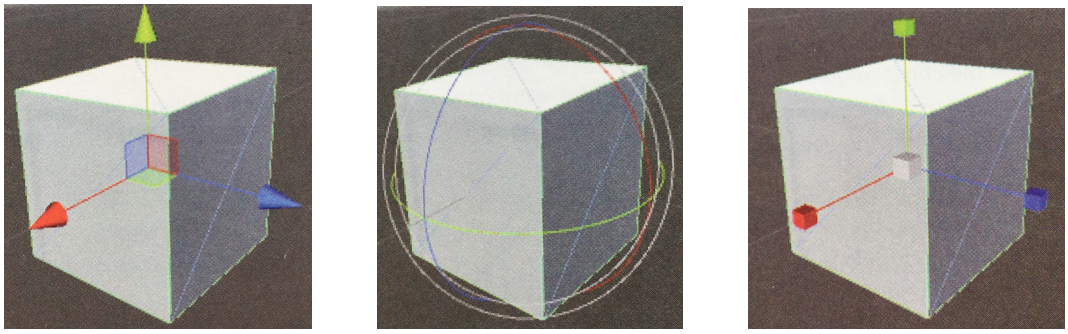


Figure 2.28: Left: gizmo to translate. In the middle: gizmo for rotation. Right: gizmo for scaling.

2.11 Mathematics of transformations

Behind transformations of objects in scenes lies a specific mathematics[8]. Each 3D virtual scene object is determined by the position of its vertices in a local three–dimensional coordinate system. Any transformation of object, such as translating or rotating, changes positions of these vertices. Consider two states: an original position of vertices before transformation and a new position after transformation.

A typical 3D transformation is represented by transformation table (matrix) which is used to recalculate vertex positions. The transformation table has form of 4 times 4, ie it is given by an ordered array of 16 numbers organized into 4 rows and 4 columns

$$\begin{pmatrix} x_{11} & x_{12} & x_{13} & x_{14} \\ x_{21} & x_{22} & x_{23} & x_{24} \\ x_{31} & x_{32} & x_{33} & x_{34} \\ x_{41} & x_{42} & x_{43} & x_{44} \end{pmatrix}.$$

Each vertex in the coordinate system is determined by three distances from the origin. Denote these distances by coordinates (x, y, z). Since you ask to include all transformations with a single organization of the transformation matrix, it must have an extra dimension. For this reason, the so–called homogeneous coordinates are used in the transformations, when the three coordinates are extended to the four so that number 1 is substituted for the last coordinate. Thus, the four coordinates $(x, y, z, 1)$ are created from the original three ones. New coordinates $(x_t, y_t, z_t, 1)$ are calculated after applying transformation (hence index t expresses transformed) by multiplying the original coordinates by transformation table

$$\begin{pmatrix} x_t \\ y_t \\ z_t \\ 1 \end{pmatrix} = \begin{pmatrix} x_{11} & x_{12} & x_{13} & x_{14} \\ x_{21} & x_{22} & x_{23} & x_{24} \\ x_{31} & x_{32} & x_{33} & x_{34} \\ x_{41} & x_{42} & x_{43} & x_{44} \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}.$$

Translation of an object vertice by selected increments Δx, Δy, Δz in individual axes is realized by the following product

$$\begin{pmatrix} x_t \\ y_t \\ z_t \\ 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & \Delta x \\ 0 & 1 & 0 & \Delta y \\ 0 & 0 & 1 & \Delta z \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}.$$

[8]This section can be omitted in reading.

You could verify that for transformed coordinates it is true that $x_t = x + \Delta x$, $y_t = y + \Delta y$, $z_t = z + \Delta z$. Scaling in individual axes given by multiplicative factors s_x, s_y, s_z is provided by the following product

$$\begin{pmatrix} x_t \\ y_t \\ z_t \\ 1 \end{pmatrix} = \begin{pmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}.$$

Indeed, you could verify that $x_t = s_x \cdot x$, $y_t = s_y \cdot y$ and $z_t = s_z \cdot z$. Rotation of vertices by angle α must be done for each axis separately, ie sequentially for axes x, y and z using products

$$\begin{pmatrix} x_t \\ y_t \\ z_t \\ 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(\alpha) & -\sin(\alpha) & 0 \\ 0 & \sin(\alpha) & \cos(\alpha) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix},$$

$$\begin{pmatrix} x_t \\ y_t \\ z_t \\ 1 \end{pmatrix} = \begin{pmatrix} \cos(\alpha) & 0 & \sin(\alpha) & 0 \\ 0 & 1 & 0 & 0 \\ -\sin(\alpha) & 0 & \cos(\alpha) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix},$$

$$\begin{pmatrix} x_t \\ y_t \\ z_t \\ 1 \end{pmatrix} = \begin{pmatrix} \cos(\alpha) & -\sin(\alpha) & 0 & 0 \\ \sin(\alpha) & \cos(\alpha) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}.$$

Alternatively, transformations are greatly simplified the use of so called *quaternions*, which represent an extension of complex numbers by additional imaginary units. For example, you could know from complex numbers, that multiplying by a complex unit i means rotating a right angle in the imaginary axis counterclockwise.

Quaternions contain three imaginary units simultaneously. Therefore, a four–dimensional system of $a + ib + jc + kd$ arises, where i, j and k represent three different roots of $\sqrt{-1}$. The three imaginary coordinates b, c, d describe rotations, the fourth a the translation. For example, multiplication by one of the complex units i, j, k means turning at right angle in the respective axis counterclockwise. In such a way, quaternions have use in virtual reality and in general 3D computer graphics. Thus, by their nature, they can be beneficial in performing three–dimensional transformations.

2.12 3D models, materials, textures and shaders

Objects that are often used in the real world are presented by models. The latter mean, for example, models of cars, boats, trains or airplanes with which you played in childhood or you are building them up to now. A basic principle of models is that they look (almost) as real patterns, however they are simplified by properties that are not essential at the time. For example, models of cars or aircraft sizes are much smaller when someone plays with them sometimes disregarding some features like door opening or wiping against rain. Similarly, you could simplify models of castles, dinosaurs, trees, etc.

Models for the virtual environment are needed. Even, very many models is needed to build virtual worlds such that model real worlds. When a 3D model is uploaded to the virtual scene, it becomes an object of the virtual world.

Virtual 3D models (generally 3D objects) are given by positions of their vertices. For simple regular surfaces such as a cube suffice to introduce their real vertices, for more complex models with irregular surfaces or peaks must be vertices substantially denser. Vertices are connected by a triangular network. Such a network of interconnected triangles is called *mesh*. Triangles connected to each other by this way can shape even complex surfaces. Triangular meshes can also be processed very quickly. Fig. 2.29 shows example of a mesh. You could see a network of interconnected triangles that is denser in more complex parts of the surface. Such a network approximates a dolphin surface[9].

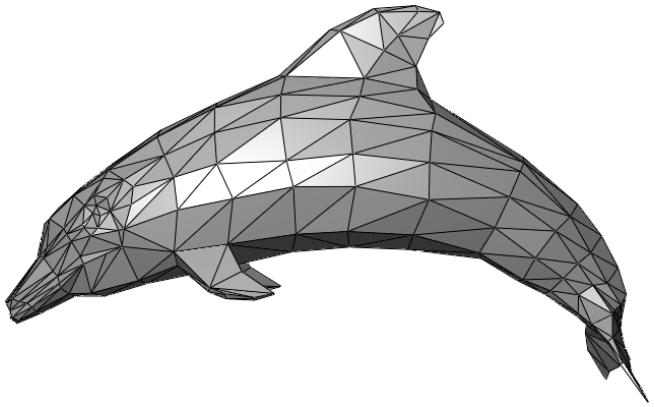


Figure 2.29: Mesh of a dolphin. Source: `https://en.wikipedia.org`

Why are interconnected triangle networks used? When processing models, an associated set of vertices is used. The fewer vertices the model has, the faster it will render. Triangles connecting vertices have the following suitable properties:

1. When asking geometric and mathematical simplicity, triangles are preferred. Their three vertices represent the minimum for defining an area. The triangle also has clearly defined normal[10].

2. When needing to make a new triangle, all you need is a single vertex. The first triangle requires 3 vertices, two 4, three 5 vertices, etc.

3. When constructing a mesh from triangles, any 3D object can be modelled.

Other polygons do not have such good properties. If more triangles are used and as they are smaller, then better the surface approximation will be. Fig. 2.30 illustrates importance of size and density of a triangular mesh. The smaller the triangles are, the denser mesh and surface and shape approximation is. It is illustrated on small and ordinary thing – walnut in Fig. 2.31.

[9]The idea of approximating a surface by a system of triangles probably comes from Archimedes (288 – 212 BC). He approximated an area of the parabolic segments, which he filled with triangles having a known area. Archimede's insight that any surface can be convincingly approximated by a triangular mesh is used, for example, in animated films such as Shrek or Finding Nemo. It was tens of thousands of triangles on Shrek's impressive ears, similar to the trumpet, just like Shrek's round stomach. In Avatar and his imaginary world named Pandora, each plant was made up of a million triangles. There were many such plants.

[10]Perpendicular to the triangle surface.

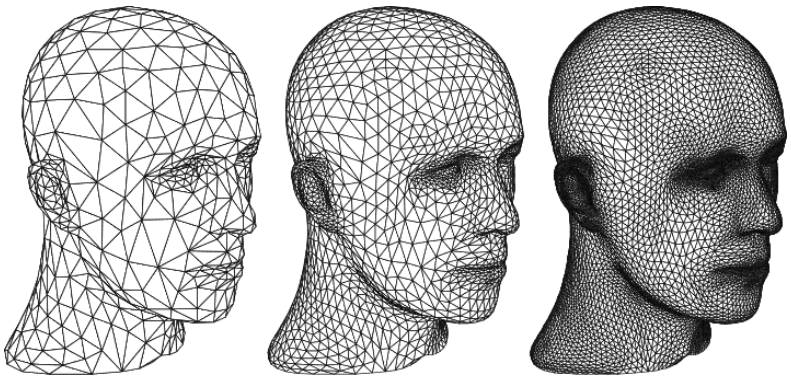


Figure 2.30: Three differently densed head meshes illustrate improved surface coverage. Source: (Zorin, Schrőder, 2000).

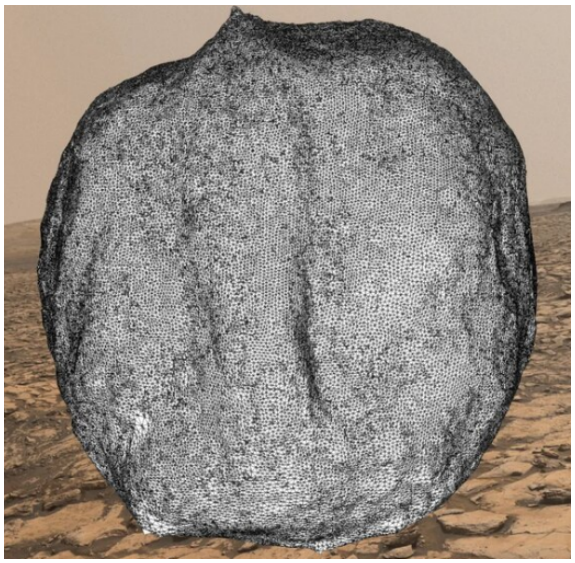


Figure 2.31: Detailed mesh covering the walnut surface.

Note that the terms model and mesh are often used interchangeably because the the model is defined by a mesh as well. However, there is a noticeable difference. Mesh contains all vertices and triangles that define the shape of a 3D object. So when describing the modeling, the mesh term is used. On the other hand, the model means an object which contains a mesh. The latter determines model surface. However, the model can also contain animations, textures, materials, shaders, etc. So if only vertex information is mentioned, the term mesh is used. If there is something extra, the term model is more appropriate.

Color triangles with a neutral gray as in Fig. 2.29. Although you get surface of a real object, it is empty and boring. In order to look real, you need to apply *material* of a model as shown in Fig. 2.32. The model material determines that, for example, the dolphin shown in Fig. 2.29 will appear as real color and its surface will be matt or shiny, respectively. At the same time it is necessary to ensure that the material can be applied quickly and cleanly without any major work.

Material of a model is determined by the following components:

1. Parameters represented by colors, numbers, matrices, or vectors.

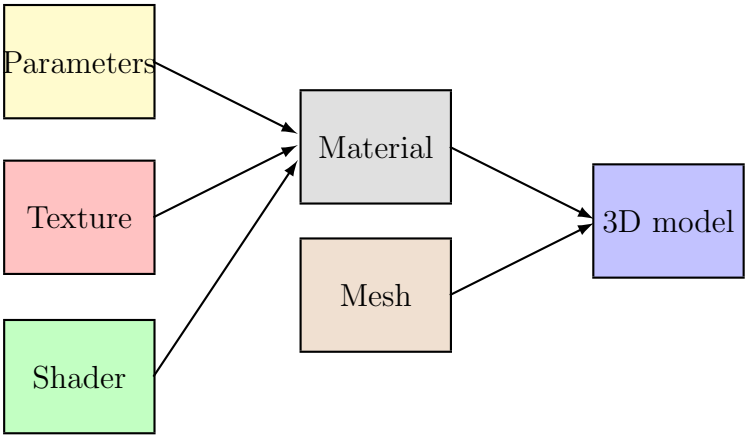


Figure 2.32: 3D model components.

2. Texture which determines what is drawn on the model surface. Textures are 2D images applied to 3D objects that change dull 3D models into colorful ones. The process is similar to a situation where a tin gray can of beef in its own juice is wrapped in a 2D colored paper to mark and describe this food. A colorful paper wrapper is like a texture. When you wrap the gray can, it will look much better on its surface. If a 3D model is more complicated, you also need accurate instructions (unwrap) on how to wrap the model with texture. The model itself does not know how to apply the texture and it usually chooses the easiest way. Therefore, it is advisable to use textures created specifically for the given model.

3. Shader is a graphics card program that calculates the resulting color of each point (pixel)[11]. Therefore, it determines how to draw on the model surface. It dictates what properties the surface will have. Whether it has a shiny or matt metallic surface, for example. For instance, if water gets on a wooden surface, its mesh and texture will remain the same, but the surface will be darker and shinier. Typical properties of shaders include

    - albedo (surface reflectivity), which defines a base color of the surface,
    - metallicity, which determines how metally surface looks,
    - smoothness, which enables to capture the surface age etc.

Notice that the creation of textures and shaders is in itself a research.

Thus, each material contains all three things: parameters, texture, and shader. It doesn't mean much more than a texture and shader container that can be applied to the surface of models. Customizing materials to the surface usually means selecting and setting appropriate shader. Fig. 2.33 shows an example of the offer of brick materials in the inventory of Neos. It shows texture of brick walls and you could see glossy or matt brick surfaces. The material tool loaded with selected brick material, which can be applied to building surfaces, is on the left.

[11]Shader also transforms vertices of the mesh.

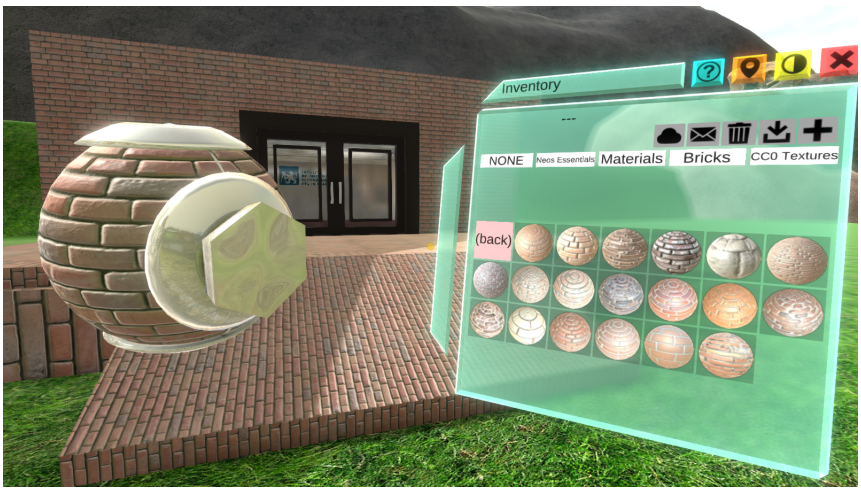


Figure 2.33: Brick materials menu in inventory.

2.13 Import of 3D models

Based on previous text you could know, that metaverse Neos includes creating tool for simple models such as cubes, cylinders, spheres, and more. For more complex models, the inventory includes a separate component called 3D models, whose content is shown in Fig. 2.34. The latter presents various built–in models for biology, lighting, furniture, nature, office, technology, etc.

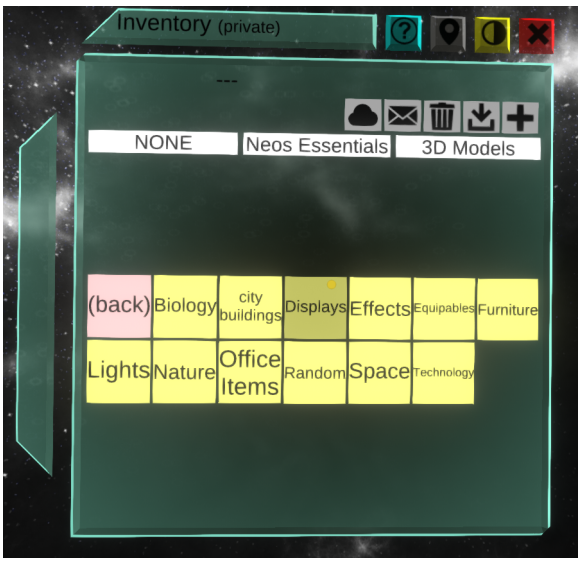


Figure 2.34: Menu of 3D models in inventory.

Built–in models may sometimes be enough. However, building of customized worlds requires 3D models created by a user, 3D models obtained from public libraries etc. Metaverse Neos provides import of such models. In the dash–board menu you apply Tools and after clicking you select File Browser. The latter window shown in Fig. 2.35 appears, presenting folders and files of the local computer on which Neos runs. Double–click with your index finger

- selets a file in the folder,

- opens the file for import.

Opening file automatically starts dialog shown in Fig. 2.35. By double–click you select 3D Model. Confirm Regular in next dialog (the model is considered as a single whole, it will not be detachable in parts). You continue through Auto Scale (automatic scaling in meters) and Import Now! The model is then imported. Using the controllers (see previous text) you can move, rotate, enlarge, reduce, duplicate, delete the imported model etc.

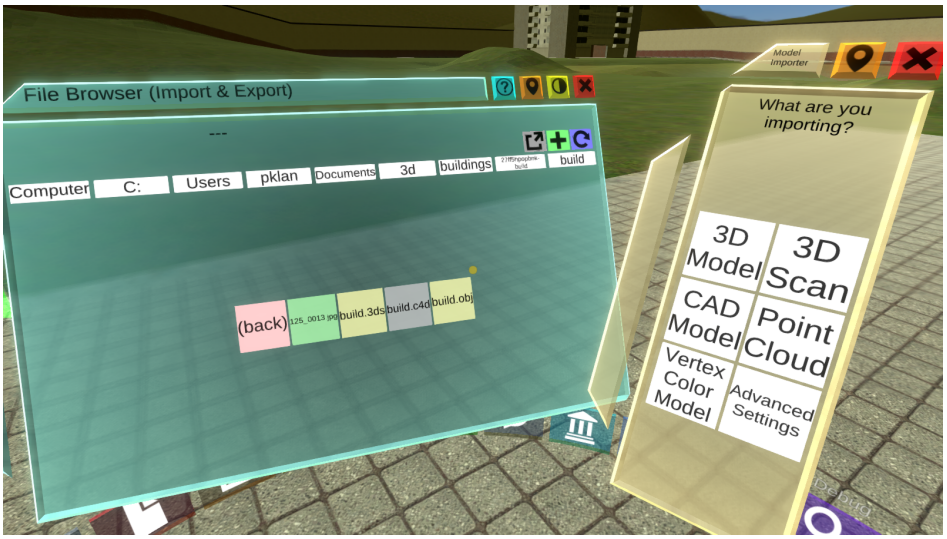


Figure 2.35: File browser window and dialog when importing models.

Supported 3D model formats use mainly .obj, .fbx and .stl extensions. In the case of another format, it is advisable to attempt to load the model. Furthermore, it is assumed that 3D models are formed by several files and formats containing textures, materials etc. A 3D model typically contains three files: .obj (mesh), .jpg (texture) and .mtl (material). Note that you could also import 2D objects such as .jpg or .png images to virtual space. On the other side, .pdf format cannot be imported.

Example 2.3 *Import of several models.* Import several models to the virtual world. Take the lunar module (NASA), the citadel (Free3D) and several custom 3D scans (roses, vase, teapot etc.).

Download files of the selected models to a computer where the Neos metaverse runs or connect, for example, a flash drive with the appropriate files. Turn on Neos and enter given world (see Fig. 2.36 left). Gradually, you continue in importing the individual models. Select Tools from the dashboard menu, open File Browser, enter the model folder, double–click the model file to open a dialog, and click the dialog to import individual models. The result of this repeated activity is shown in the Fig. 2.36 right.

Example 2.4 *Virtual lecture.* Realize a virtual reality based lecture on a topic similar to the powerpoint script via single slides.

You download individual slides of the presentation in .jpg or .png format, such as screenshots. You gradually present them, later in this text you will see that with the help of visual programming it is possible to make a projection screen on which you gradually place the slide images. A virtual lecture can be given to a virtual audience and at the same time a real one, as was the lecture called Pythagoras and Easter, as shown in Fig. 2.37.



Figure 2.36: Left: Scene before importing models. Right: Scene after models are imported.

Probably the most interesting option is to import your own models. You can get 3D models in the following ways:

a) Modeling in a specific software (Sketchup, Blender, etc.).

b) Using a 3D camera scanner (for example, Structure Sensor in Fig. **??**. An example of how to do this is at `https://www.youtube.com/watch?v=KP1_RVGXh1U`.

c) Application of so–called photogrammetry. This presents a way to systematically take photos of an object. Take a large set (preferably several tens to hundreds) of photos according to complexity of the object. Photos are then processed using special software (eg Agisoft Metashape). An example of a photogrammetry procedure can be found at `https://www.youtube.com/watch?v=IB-PCb5RMOE`.

2.14 Terrains

Virtual worlds include, to a greater or lesser extent, outdoor imaginary or real environment. The latter typically includes a terrain. Terrains mean any part of scene that represents the outdoor landscape. Terrain examples are given by snowy mountains, grassy plains, bubbling swamps, deep pools, gentle hills, sandy beaches, river valleys, sand dunes etc. Fig. 2.38 illustrates a plain grassy hilly terrain (left) and a deep canyon (right).

There are two ways to work with terrains in Neos:

1. Inventory tools. Tools will help with terrain creation. For example, in the Essential Tools → Brushes folder, there are specific Rock Brushes and Grass and Foliage Brushes. Their output looks realistic. In the 3D Models → Effects folder, you find a 3D model of water surface or 3D models of trees used in Fig. 2.20 right. In Fig. 2.39 left you could see a collection of stones obtained using various brushes from the Rock Brushes folder and a sample of a water surface (right). Another way to create outdoor landscapes is to apply different types of sky that can be found in Neos Essentials → Skyboxes folder.

2. 3D models. Terrains can be created in special programs (eg Blender). There are collections of freely available terrains as 3D models that you can import into the virtual world. There are Free 3D Terrain Models `https://www.turbosquid.com/Search/3D-Models/free/terrain` terrain collections or World Machine: The 3D Terrain Generator `https://www.world-machine.com/` for example.

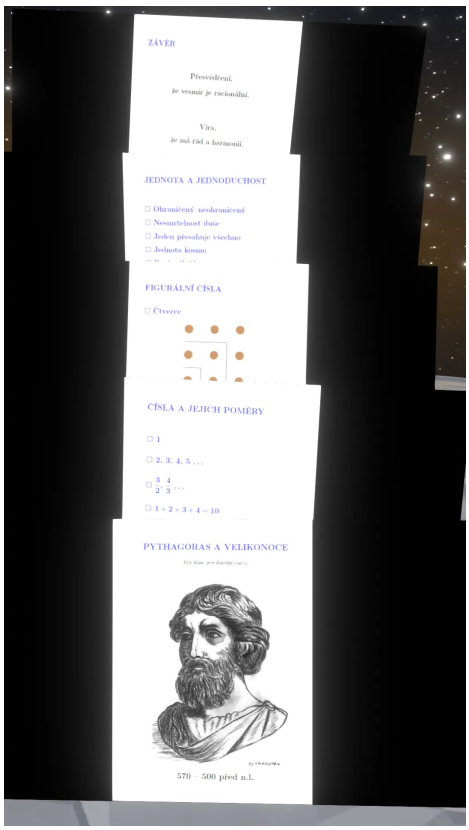


Figure 2.37: A virtual lecture on Pythagoras.

2.15 Particle Systems

Particle systems represent objects that emit other objects – *particles*. Particles can be fast, slow, small, large, flat, shaped etc. An original article by W. T. Reeves, 1983, is available in the references at the end of this book. Particle systems in virtual worlds are used for so–called *particle effects* such as flash firing, light path, fire burning, rising smoke, lightning, rain falling, fog, snow etc. If you enter keyword *particle systems* and look at image search results, you could see a number of various effects produced by particle systems, as shown in 2.40.

Working with particle systems in the Neos metaverse is for example included in worlds as shown:

a) Within. A Particle Show, where the spatial particle effects are accompanied by relaxing music.

b) Particle Setup Workspace where you can practically test a full range of particle system applications and settings.

In Neos Essentials inventory → Gadgets → Weapons you could find a series of weapons with light effects created by particle systems. To test a weapon, you point a gun by the controller beam and click the index finger to select it. The following finger double–click causes the weapon physically appears next to the directory. With a double press of the side button of the controller, you grab it. Most weapons are activated by a single click of the index finger.

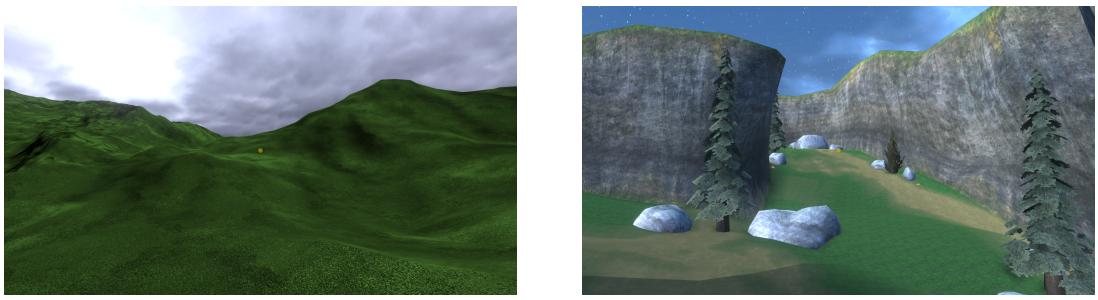


Figure 2.38: Left: hilly terrain. Right: deep canyon.

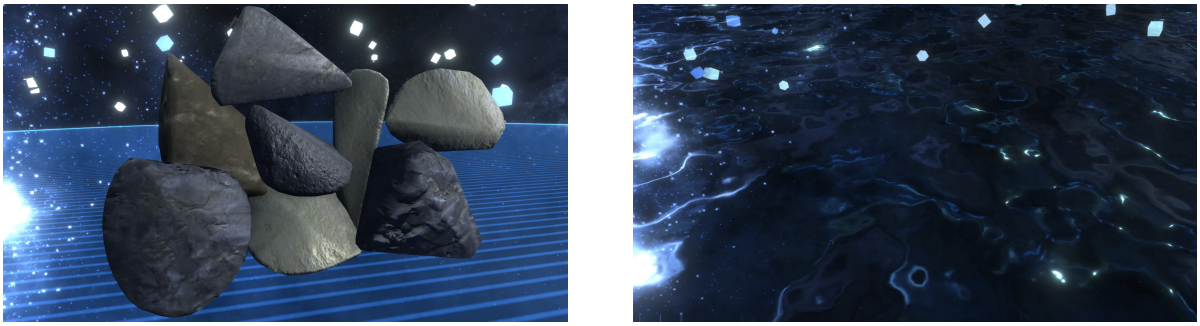


Figure 2.39: Left: collection of stones. Right: water level.

Particle systems emit particles. You could see particles as points in a coordinate system that, at any moment, have position, speed, direction, durability and color. Positions are recalculated in very short time increments called dt, typically $dt = 10$ms. Increments of each particle path are calculated from the particle velocity direction vector as

$$\overrightarrow{ds} = \overrightarrow{v}\,dt$$

and at the same time lifetime of the particle T is decreased according to the relationship

$$T = T - dt.$$

Various directional velocity vectors for different particles are shown in Fig. 2.41. The situation shows state before increment of dt (particle positions at the start of arrows) and after increment of dt, when particles move to the end of arrows. If a particle expires, the particle will perish.

Since the particle emitter can emit very many particles and very quickly, it is important that they are made in the simplest possible way and that their processing does not require excessive computing power. Therefore, 2D bounded areas are most often used, which then gives an illusion that they are 3D dimensional. When observing effects with a multitude of particles, it is not expected to look at the particles under a microscope where they can be seen straight and flat. Fig. 2.42 shows an example of such a bounded area where there is a point in the middle of the area. Particles have material (texture, shader) so that the bounded area represents a billboard. Instead of a simple point as in Fig. 2.42, billboards represent snowfalls, drops of water (rain), stars (light effect) etc. They can be used on billboards depending on the purpose.

Particle emitter, ie the source from which the particles emerge, can be thought of as the barrel of a cannon in Fig. 2.43. The dots represent particles that continuously leave the barrel with

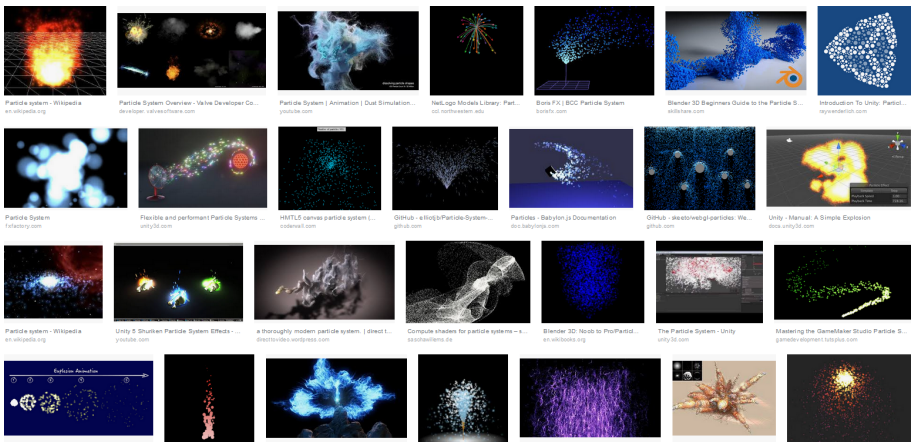


Figure 2.40: Examples of particle effects.

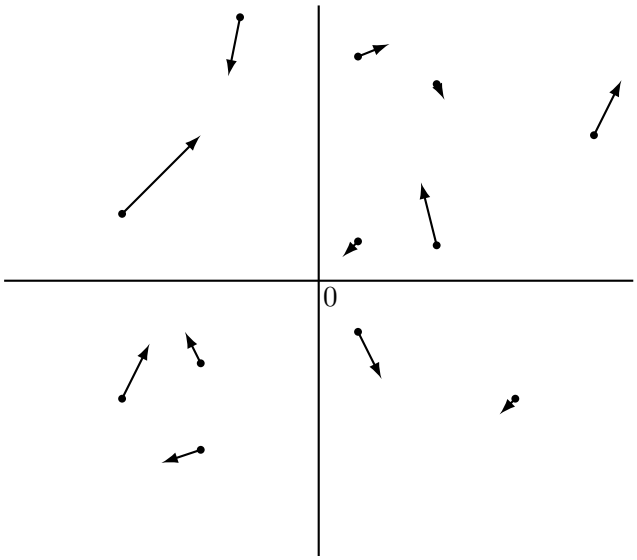


Figure 2.41: Calculation of the particle path increment.

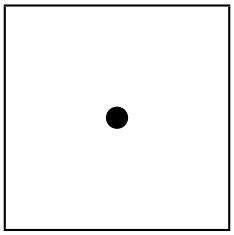


Figure 2.42: Bounded particle area.

an initial life, speed and color. Emitter characteristics determines number of particles/second (rate) or the number of explosions/second (bursts). Particles can have gravitational effects and move along ballistic curves.

Particle systems also include collisions, which are useful for all kinds of particle effects. There are mutual collisions of particles or collisions of particles with surfaces of other objects.

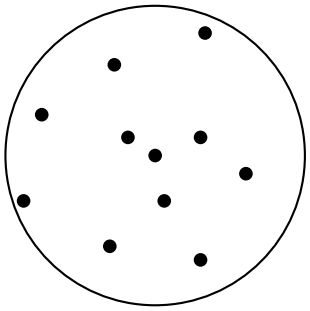


Figure 2.43: Particle emitter.

Surfaces have normal vectors according to which the particles are bounced off (see Fig. 2.44). Collisions with surfaces can also reduce life of particles.

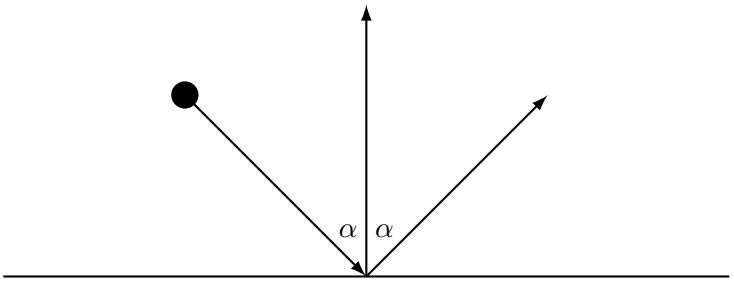


Figure 2.44: Collision of particle with surface.

For instance, pouring coffee from a coffee machine or lemonade from a can can be considered as a particle effect. In addition, to achieve visually attractive effects such as smoke or fire, you usually combine several particle systems together. For this part, you could settle with the world of Particle Setup Workspace (see Fig. 2.45), where it is easy to realize a wide range of particle effects and copy them to your scene. In Fig. 2.45 you can see an instrument panel for changing parameters of a particle system, the particle emitter is above the plate and you insert particle textures from the predefined texture panel into the empty rectangle on the right. In a moment you can create for example rainy weather, rainy wind weather, but also the environment of falling colored leaves or stars.

In the Neos metaverse, it is possible to create functional particle systems too and to set the emitter and particle parameters for wide area of particle effect requirements. Before this, however, you need to maintain the so–called scene inspector. Therefore, the particle systems will be continued in the Chapter of visual programming LogiX.

Example 2.5 *Snow blizzard, sandstorm, leaf fall, fish fall from the sky.* In the virtual world of Particle Setup Workspace, realize a snow blizzard, a sandstorm, fall leaves and fish fall from the sky.

On the left in the virtual world of Fig. 2.45 you could see the instrument panel for controlling the particle system, in the middle the particle emitter and on the right the particle menu board. For example, if you select one of the pattern leaves and press the side button to grab it, you transfer the leave, and paste it into the box next to the emitter, the leaves begin to fly out of the emitter as in 2.46. It is nice to see billboards with individual particles (in this case leaves) and also the basic principle of billboards. They are always positioned where

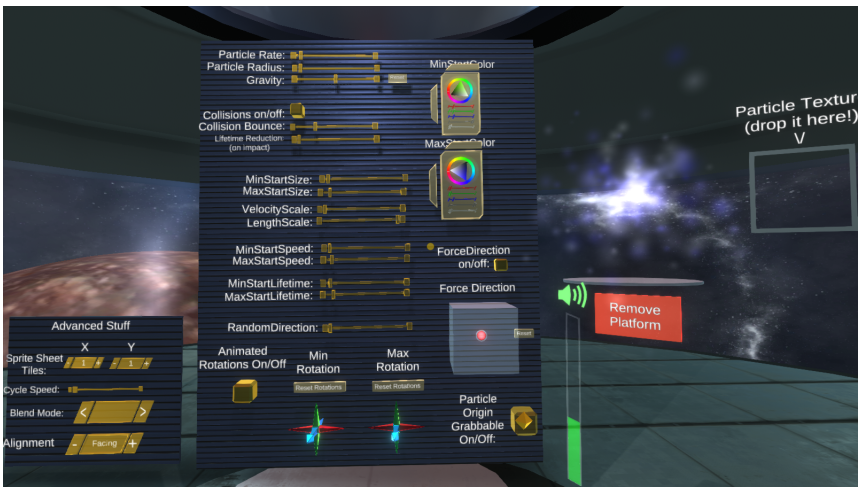


Figure 2.45: World for experimental work with particle systems.

it is important (like on camera). The instrument panel for controlling the particle system is visible to the left in the background.

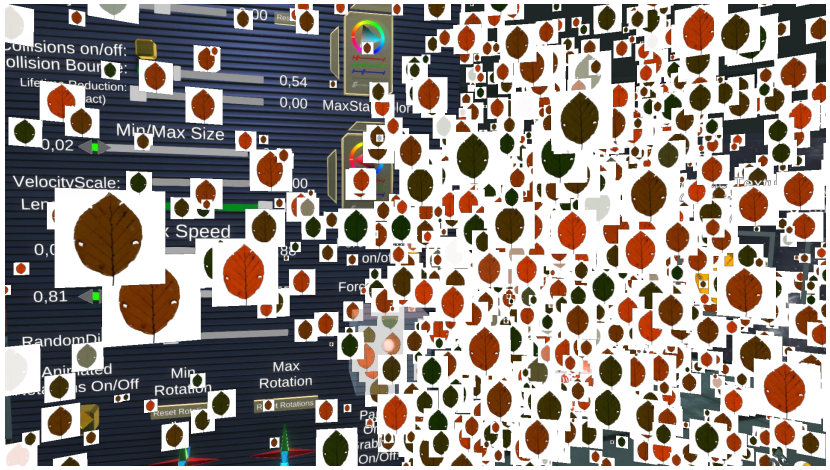


Figure 2.46: Falling colored leaves.

2.16 Lighting, photography and streaming scenes

Most virtual scenes aim for realistic effects. Therefore, they use at least one light. A use of the socalled point light is typical. The tool for its creation is found in the Essential Tools inventory (see Fig. 2.10, in the first row). Grab the tool and create a point light with a single click of the index finger. Fig. 2.47 shows a group of four point lights showing a light effect. The point light works in On/Off mode. If you focus controller beam on a point light, then by clicking the index finger you turn off the light or turn it on. Point lights can then be spread along the scene.

Point light is an ideal spot from which the light rays propagate in all directions in a straight

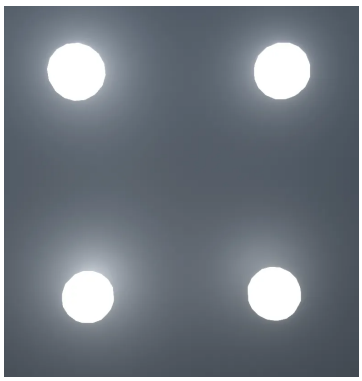


Figure 2.47: Light effect of a group of point lights.

line, similar to that shown in Fig. 2.48. If light falls on the surface of an object, it is reflected by the optical law of reflection. The intensity of the reflected light is determined by the normal vector of the surface and by the material property. A ray of light that strikes the surface perpendicularly reflects most back in the same direction. The intensity of the reflected light determines the scalar product[12] of light beam direction and the normal vector. In the case of diffuse material, if the beam is perpendicular (the angle with normal is zero), the scalar product becomes maximal, since $\cos(0) = 1$. In Fig. 2.48, the intensity of the light beam reflected in the same direction as the beam falls will be smaller than it would be in the case of a perpendicular reflection.

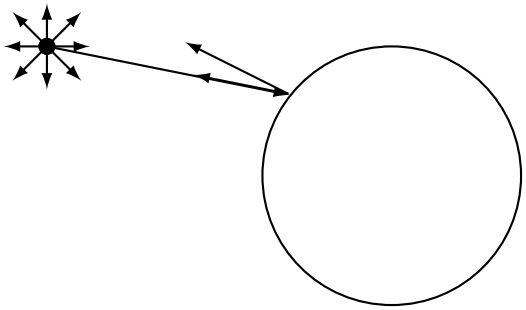


Figure 2.48: Effect of point light.

More lights can be found in the the Neos Essentials → 3D Models folder. There is a subfolder named Lights, where you find 3D models of a flashlight (cone light), burning torches and lighted kerosene lamp. You will discover other lighting options later when working with the scene inspector.

Another frequent requirement is to take a photo in scenes. A camera is available in the Neos Essentials inventory, in folder Camera, for these cases (see Fig. 2.49). By a double–click of index finger the camera physically appears in the scene. By pressing the side button of the controller, the camera can be moved and rotated as needed. The back screen shows the scene being photographed. Keeping the camera and clicking the index finger will immediately start the camera to take a photo. The photo appears next to the camera.

[12]The scalar product of two vectors $\overrightarrow{x} \cdot \overrightarrow{y}$ is determined by the product $|\overrightarrow{x}||\overrightarrow{y}|\cos(\alpha)$, where α represents the angle between the vectors and $|.|$ indicates size of the vector.

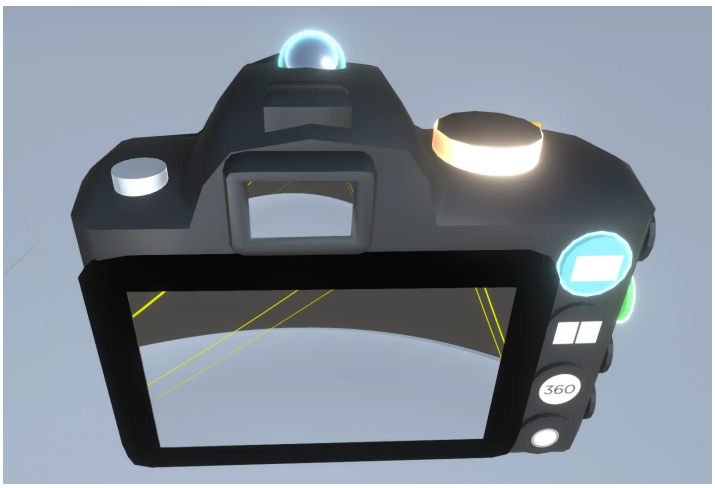


Figure 2.49: Camera in the virtual world.

To save photos to a computer device, select Tools → File Browser from the dash–board menu. Select the directory and save the photo by grabbing it by pressing the side button and using the other controller activate + at the top right of the opened file browser window. A simple dialog form will appear, where you enter the file name and activate the related button to save the file. The file appears between the files displayed by the file browser.

A welcome option is to either shoot a video of the scene or to stream it. Metaverse Neos provides an interactive camera for this purpose, which can be found in the dashboard Tools → Camera/Streaming menu. The camera is controlled via control panel that appears at the same time as the camera (see Fig. 2.50). The camera allows recording video from a scene or streaming video to registered users such as Twitch `https://www.twitch.tv` streaming platform. Neos channel in Twitch is at `https://twitch.tv/NeosVR`. Video or streaming[13] starts when you activate Mirror to display button in the camera control window. An example of a streaming scene is shown in Fig. 2.51.

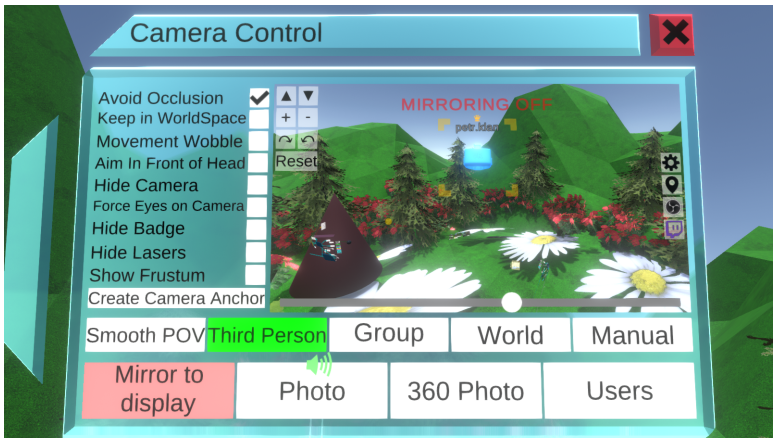


Figure 2.50: Interactive camera for video and streaming.

[13]In order to stream, an external OBS (Open Broadcaster Software) software is needed to install.



Figure 2.51: A streaming scene example (15. 11. 2019).

2.17 Scene inspector

Inspector represents one of the most important metaverse element. It allows you to view and change all properties of virtual scene objects. You select an object from the virtual scene hierarchy, the inspector opens its properties, and when you change an object property, it automatically applies the change. When you can change everything in the virtual scene, you get enormous creative power.

Given the importance of the scene inspector again: the inspector enables you view and change all properties of a selected object. The inspector has its own tool in the inventory, in the Essential Tools folder. The instrument is black with two transverse yellow stripes (see Fig. 2.10, penultimate row). Select the tool with a controller beam, confirm with a simple click of the index finger and double–click physically place it next to the tool folder. Now you can grab the tool by pressing the controller side button twice and release it when you do not need it.

To open the scene inspector window, use your thumb to press the knob of controller menu and select the Open Inspector icon and double–click with your index finger.

Show it in an example. Create an empty world from the New World dashboard menu as shown in Fig. 1.15 (Basic Empty). Import a 3D model to this world: from inventory of 3D Models folder or own. Fig. 2.52 shows a simple 3D model of a rose. Now you could grab the Inspector tool, press the menu knob of the controller with your thumb, select Open Inspector icon, and double–click the index finger on the controller. Scene inspector opens a window as shown in Fig. 2.52 left.

Looking at contents of the window, it has a two–column arrangement. In the left column, you could see a hierarchical list of scene objects. For rose model there you could see item Model.obj. On the right it is space for components and object properties. It remains empty so far.

If you focus the controller beam on any object in the left column, in our case Model.obj, and double–click with the index finger, the right side of the window fills. You could see boxes with current positions, rotations, and scale of the object in each x, y, z axis. If you want to try what the change of a value will bring, you focus the controller beam on the appropriate box. After a simple click of the index finger, the keyboard appears, where the focusing the controller beam to a key followed by a simple click enter the new character. The latter is immediately transferred to the selected box.

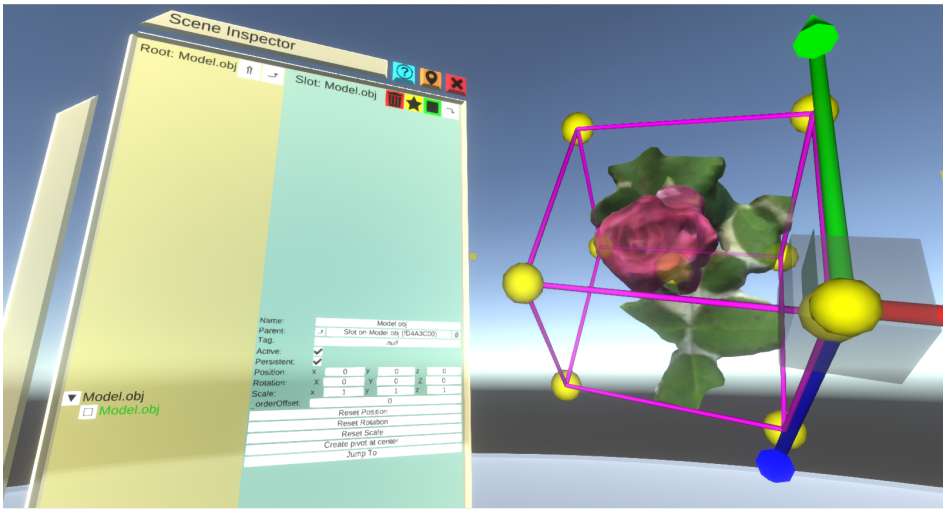


Figure 2.52: Inspector window.

Yet another significant change could be noted. Once you have selected scene object in the left column, a gizmo appeared around that object (see Fig. **??** right). Recall that you can also rotate an object, move or scale it by pulling the gizmo axes. Focus the beam on the axis, press the index finger and pull the controller in desired direction. Above the object there is a gizmo menu (see Fig. 2.53), where after focusing on and pressing the index finger, you could switch between gizmos for shift, rotation or scale.

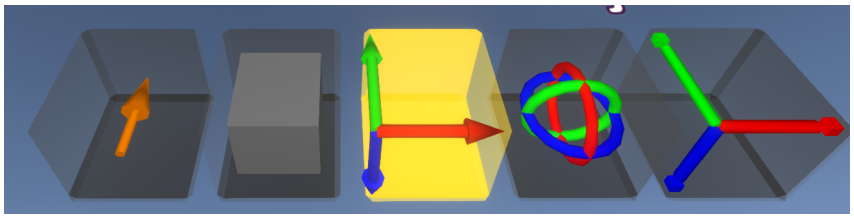


Figure 2.53: Basic menu of gizmo.

To cancel Gizmo on a selected object, use your thumb to press the menu knob of the controller and choose the Deselect All icon and click once with your index finger.

The Attach Component button is visible in the right column of the inspector window. You can add a number of other components to the selected object. For example, if you add the Spinner component and enter velocity value in one of the axes, the object begins to rotate. If you add a component, the component appears in the right column of the inspector window. Try to progress through Attach Component → Transform → Driver → Spinner.

All the essential aspects of working with the inspector result from this simple example of 3D rose model. So if you request to change something in the scene, you usually find help using an inspector. Application of the latter can be summarized in the following steps:

a) Grab the inspector tool. The tool can be found in the inventory in the Essential Tools folder.

b) Open the Inspector window from the controller menu by double–clicking Open Inspector. The window is a two–column as shown in 2.54. In the left column, there are

objects in a hierarchical arrangement of the scene (parents and their children). When the selected object is activated, its components and properties appear in the right column.

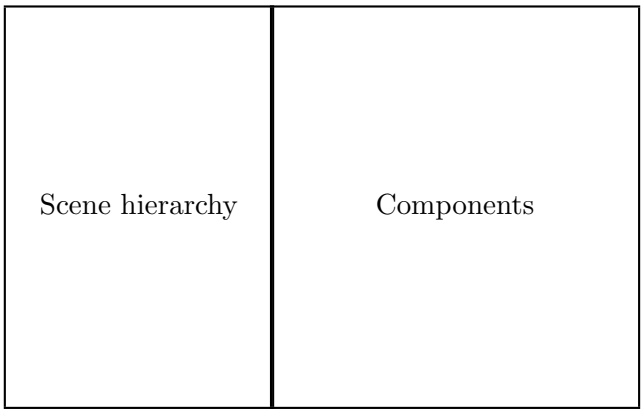


Figure 2.54: Schematic arrangement of the inspector window.

c) Some properties, such as position, albedo color etc., can be changed numerically, many components and properties are turned in a logical on/off format. For example, a property called *persistence*, or the permanent presence of the object in a scene, is very important. If you disable persistence, the object will be missing the next time the world loads in the scene.

d) Via Attach Component, you could add a number of properties to objects. Objects can rotate (Spinner), twist (Wiggler), wobble (Wobbler), for example in the wind, they can do it slowly (SmoothTransform), surfaces can gravitationally attract (GrabbableReceiverSurface), objects can play audio (Audio) etc. Usually, you go through the menu Attach Component → Transform → Drivers. Parameters of rotation, wobble etc. are entered numerically. You learn best by trial and error method. For example, you could use Spinner component to rotate the wind power plant rotor in Fig. 2.55 by entering a non–zero velocity in the appropriate axis.

Example 2.6 *Windy weather.* Use a few inventory trees to build a grove and simulate windy weather.

The grove with trees can be arranged, for example, as shown in Fig. 2.20 right. Gradually select trees (or duplicate them) from the inventory. When you open the scene inspector, you add a Wobbler component (Attach Component → Transform → Driver → Wobbler) to each tree, which simulates an irregular rocking of an object in windy weather. The same can be done with fire flames in the foreground. When you look at the scene, the simulation of windy weather will look realistic.

2.18 New objects

If you take the inspector tool in hand, the path to some new virtual scene objects opens in addition to these included in inspector. This is especially about Create new... When you press the controller menu knob with your thumb, in addition to the Open Inspector, you will

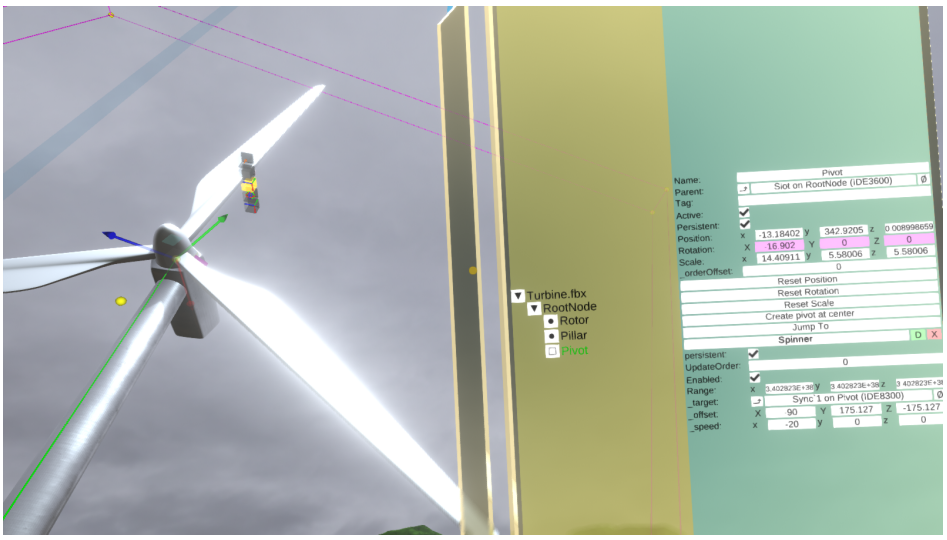


Figure 2.55: Wind rotor spinning component.

also get the menu Create new..., which you could select with one index finger click. Among other things, there will be proposals for objects that can be newly created and placed in a virtual scene (see Fig. 2.56):

a) 3D Model: create models of types such as cylinder, cube, quad, sphere, classic toroid (torus) etc.

b) Light: create

- point light,
- directional light,
- spot light,

for use in lighting the scene or some parts of it. The principle of point light was already met in Fig. 2.48. Fig. 2.57 shows the directional light principle. The light is directed straight at the object from one direction. Because of their distance, the rays of the sun falling on the Earth's surface can be considered as directional light. Spot light can be encountered in the light of inventory object – flashlight. The same type of light can be used for car headlights. It is a light beam that extends from a point at some limited and usually acute angle, for example, 45°.

c) Materials: creates material.

d) Object: create objects such as Avatar Creator, Camera, Mirror, Video Player etc.

e) Empty Object: creates an empty object. An empty object can be useful when gluing several parts into one object (eg propeller blades). Copy the glued object, including all parts, to the new object. This gives one gizmo and easier manipulation with the glued object (eg rotation).

f) Particle System: creates empty particle system as a separate object. Using the inspector you could set the system to meet given requirements for position, amount of particles, color, speed, size, scattering angle etc. The situation after creation of particle system, the change in the number of particles and their color is shown in Fig. 2.58.

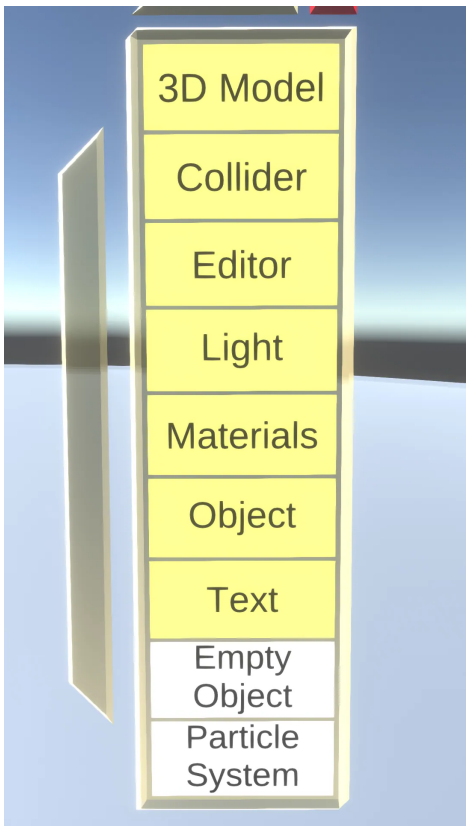


Figure 2.56: Menu to create new objects.

Example 2.7 *Pouring drink from can.* Pour the drink from the can into a cup.

Download a 3D model of the can and import the model. Tilt the can and use the particle system with the particle emitter at the point of outflow from the can to pour the drink. You could create a new particle system, use the gizmo to move its origin (emitter) to the outlet, and adjust the particle system parameters to visually resemble the flow from the can to the lower located cup. The cup model can be downloaded and imported, too. In terms of selecting particle parameters, it is advisable not to be afraid of trial and error method. There are many parameters of the particle system in the inspector, but you often could estimate the meaning of the names.

2.19 Customized materials

In this section, a way how to adapt materials to creative intentions will be presented. The materials determine:

1. How light will interact with object surfaces.
2. How object surfaces are rendered.

The physics of color vision is defined by the following process. When you look at the colored surface, you see a color which is determined by photons reflected from the surface. The photon of light falls on an atom of the surface. The atom absorbs energy of the photon and uses it to send a photon to the eye. Its energy determines wavelength of light, in other words

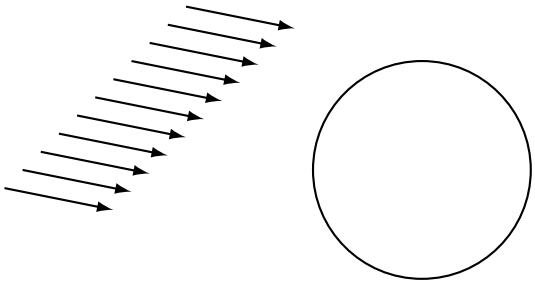


Figure 2.57: Directional light principle.

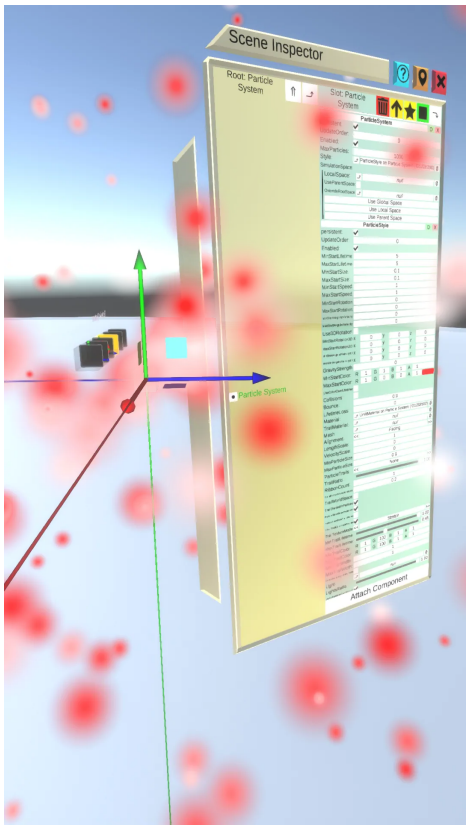


Figure 2.58: Created and modified particle system.

its color. Each color corresponds to a different photon energy.

Fig. 2.32 shows that each material combines shader and texture. While the shader, as a program on the graphics card, defines a view of the surface (its output determines color of each pixel), textures provide surface details.

The aim of virtual reality is to look surfaces of objects, or their materials, as realistic as possible. The basis of this view is the physically conceived interaction of light with the surface. Therefore, materials are referred to as *physically–based* (either PBS – Physically Based Shading or PBR – Physically Based Rendering), which, in other words, indicates that physical reality is approached in terms of interaction with light. An article on the basics of light behavior on surfaces of J. Russell is available in the references at the end of this book. According to the surface behavior of the material (the ratio of the amount of light reflected from the surface and the amount of light that hit the surface), two types of light can be

distinguished:

1. reflected or specular,
2. diffused.

Specular light indicates that the light beam strikes the surface and is reflected by the refracting law at the same angle symmetrically to the normal vector (Fig. 2.59). Similarly, a soccer ball bounces off the surface. For example, metal polished surfaces reflect light almost perfectly. The mirror also provides a perfectly smooth surface.

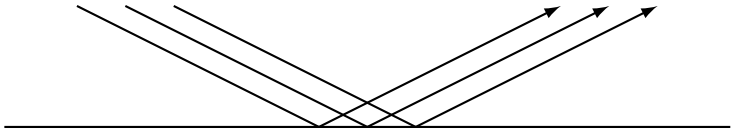


Figure 2.59: Specular light on a smooth surface.

However, not all light may be reflected from some surfaces. Diffused light occurs when light rays penetrate below the surface into the interior of an object. There is either absorbed by material (usually converted to heat) or diffused. Some of the latter radiate out as in the Fig. 2.60. Absorption and diffusion vary at different wavelengths, which adds color to the surfaces. For example, if an object diffuses in green color under the surface and the green radiates out, then it will look green, although it may not be so intense. Colors of diffused light surfaces are therefore duller, less pronounced, because they do not reflect so much light.

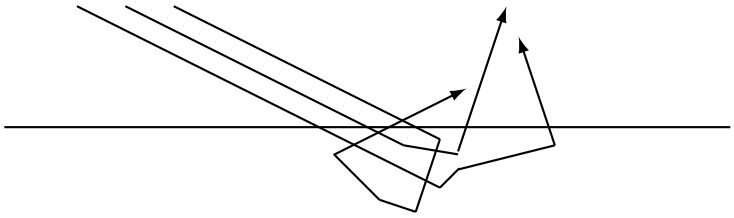


Figure 2.60: Diffused light on a smooth surface.

Most real world surfaces have very microscopic imperfections: small grooves, cracks and lumps. Although invisible to the eye, these microscopic elements have a significant impact on reflection of light from the surface (diffusion does not suffer). Normal vectors of such a surface are not parallel as in case of a perfectly smooth surfaces. Each light beam strikes a part of the surface with a different normal orientation, so the angles of the reflected beams very differ. An analogy is when the ball bounces at an unpredictable angle. The thicker surface, the more reflected light becomes blurry.

Use the inspector tooltip in the Essential Tools inventory folder to create materials. Grab the inspector tool, press the controller menu knob with your thumb, select Create new... and confirm it with one index finger click. The menu shown in Fig. 2.56 appears. Use the controller beam to select Materials and confirm it with one index finger click. From the menu you receive immediately, you can select:

- PBS: includes a wide range of materials such as metallic materials (metallic–like materials), speculative materials, emissive materials (shining surface materials), Rim Color materials (emissive edge materials) etc.
- PBS Metallic.
- PBS Specular.

The latter two PBS Metallic and PBS Specular differ in their inputs, but basically with them you can achieve the same thing: diffusion, smoothness or metallic look. PBS Metallic is generally easier to work with.

As soon as the material is selected, a material window will appear and next to it a material ball will automatically reflect all the changes you make in the material window. Fig. 2.61 shows an example of material window. Material labeled PBS RIM Metallic was selected from the PBS menu. The Albedo color (AlbedoColor) was set to $(0.8, 0.8, 0.2, 1)$ and the border color (RimColor) was set to $(1, 0, 0, 1)$.

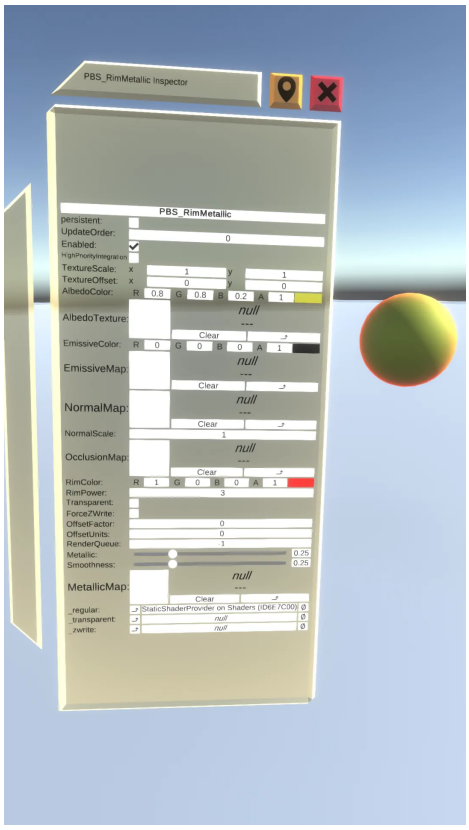


Figure 2.61: Material window with material ball.

Material ball is used as a charge for the surface material tool in Fig. 2.15 right. If you create own material, you are not limited by the supply of materials from inventory. Therefore, if you charge material tool with the material shown in Fig. 2.61, the new material can be applied. Recall that you get the material tool in the Essential Tools folder. With one hand you grab the material tool (by double pressing the side button of the controller). With the other controller you focus on the material ball and by pressing the side button you grab it and insert it into the material tool. There you release the material ball. Now for example, if you focus the material tool on the 3D model of a car and click with your index finger, the new material is applied to the car surface as in 2.62.

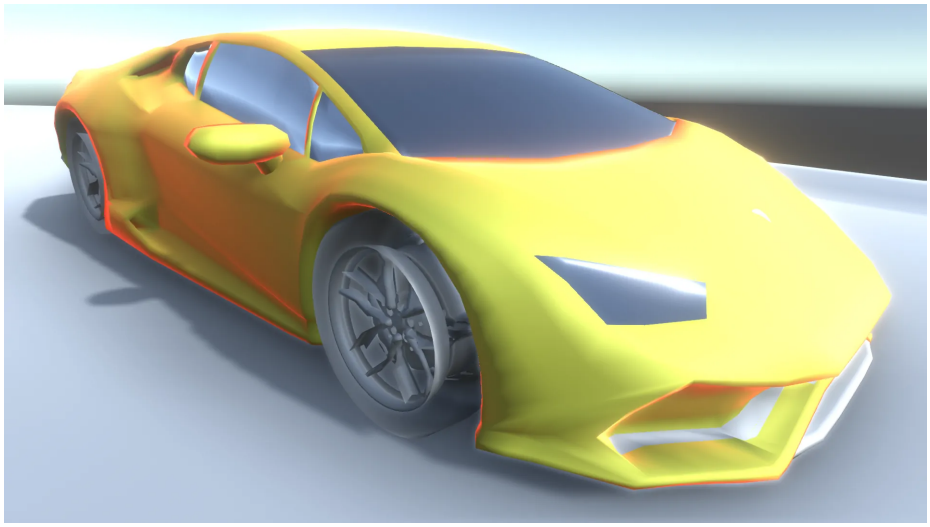


Figure 2.62: Application of material to the car model.

You can use some transparent material for the car window glasses. You select such material in the material window by selecting Transparent in the BlendMode box. As a headlight material, any material that emits (is emissive) can be used.

When you look at virtual worlds and discover that some material is the right material (it can be a photo), you can get the material if you you focus on the material by the material tool. Pressing the thumb on the large secondary action knob causes that the material from the marked surface is replicated to the material tool. This material can then be used like any other material, removed from the material tool, stored in the inventory etc.

In conclusion, creating illusions of surface irregularities in order to imitate real surfaces is a complicated problem in itself. So–called normal maps are sometimes used to achieve the illusion of surface irregularities, where a normal vector is assigned to each pixel of the surface. This then affects the calculation of the surface illumination, for example by the method of Fig. 2.59 for specular light.

Example 2.8 *Colored car.* Import a 3D car model and color it with your materials.

You import 3D car model. Take the inspector tool, open Create new..., select Materials and choose PBS Metallic. In the material window (again using trial and error) you set RGBA of the material. You can also set the metallic appearance. You get a material tool from the inventory and charge it with just the right color. Then you select transparent material in BlendMode. In the case of darkness, you can place spotlights in the reflectors.

Example 2.9 *Face as the form of material.* On the white ball, print the face of the Pythagoras known from the right–side triangle theorem.

From the Essential Tools inventory, you take a shape tool and shape a sphere that you size appropriately. You look up an image portrait of Pythagoras you import. You take the material tool from the same inventory, focus the beam on the Pythagoras image and press the big knob with your thumb. The material tool then replicates the image material. The respective material ball appears in its stack. Focus the beam of the material tool on the ball and click with the index finger. The material appears on the ball. In the inspector, the ball material can be further modified as shown in 2.63. It shows Pythagoras face printed on a sphere (left), at the top right is a material ball created by replicating the material from the

image, and at the bottom right is the image that served to replicate the material.



Figure 2.63: Pythagoras face printed on a ball.

2.20 Saving and exporting objects

Each virtual scene object (2D image, 3D model, avatar, world, material, visual programming scheme – blueprint etc.) can be stored in inventory. Only registered users can save. To login to Neos metaverse, use Login in the dash menu. Registration is simple, you fill in only username and password. It runs without any charge.



Figure 2.64: User inventory.

When saving to inventory, open inventory on the enter page (see Fig. 2.64). You will see the stored objects. If you want to save another object, you focus on the object by the controller beam and hold it by pressing the side button of the controller. Focus the controller in the other hand on + sign at the top right of the inventory window and click with your index finger. Shortly thereafter, the object appears among the inventory items.

Stored objects can be deleted from the inventory, just focus on the object by the controller beam and click with the index finger. The selected object is highlighted. Then you move the beam to the trash icon and click the index finger to specify the object to delete. Click the index finger repeatedly to confirm the object is deleted.

Just as you store objects in inventory, you can export objects to computer storage devices. To do this, use the Tools → File Browser in the dashboard menu. A file browser window will open where you can use your index finger to click through to the device where you want to save the object. Fig. 2.65 shows an example of the file browser window for the external flash disc F:.

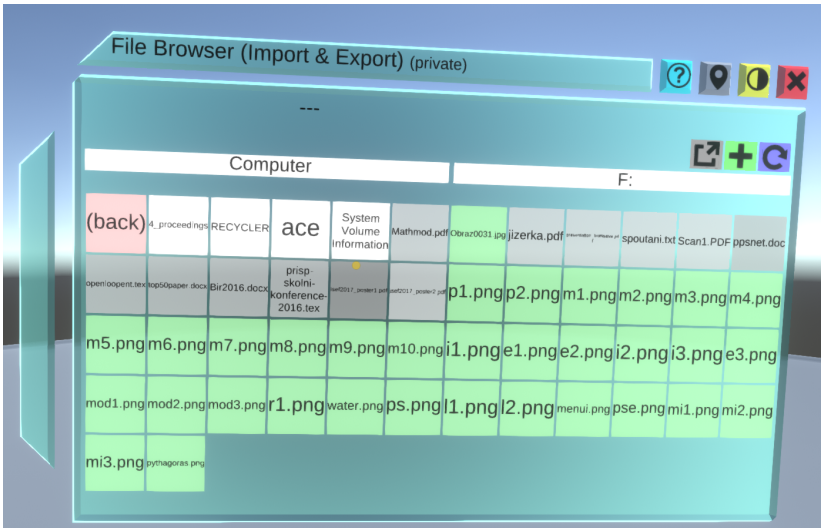


Figure 2.65: File browser window.

You see a list of files. When you want to save an object, you proceed in the same way as when saving an object to inventory. If you want to save the object, you focus on the object by the controller beam. To grab the object, press the side button of the controller. Focus the second controller beam on + sign at the top right of the inventory window and click with your index finger. A simple form appears, where you enter the name of the related file and check the format. Activate the export button with your index finger. The object will appear among the saved files shortly.

Chapter 3

AVATARS



Figure 3.1: A naive avatar. Source: unknown author

An avatar is a creature that represents user in the virtual environment. The chapter deals with:

- Social relationships and experiences of avatars.
- Appearance and creation of avatars.
- Movements and animations of avatars.
- Presence of avatars in virtual scenes.

Fig. 3.1 illustrates the fact that an avatar can be the source of many individual imaginations. From concept of a worship or individual self–presentation to an empathic social sharing. From the notion of supernatural perfection to a naive creature.

3.1 Social relationships and experiences of avatars

While in the real environment you are meeting human beings, in the virtual environment you present yourselve as an avatar. Avatar is a character that you embody in the virtual world. The other residents of virtual world see you as the avatar. You can look as a real creature or almost any way (for example bread, monster, sausage, car). Avatars have, to date, often used anthropomorphic features

1. the body of a human being,
2. the head of an animal,

made as animes – cartoon characters based on the Japanese animated movies or as types of robots.

The term avatar comes from ancient Indian culture (Hinduism) and refers to the case when a worshiped human being returns (descends) after his earthly departure and takes on some other appearance, such as a human figure with a goat's face as shown in Fig. 3.2.



Figure 3.2: Avatar with goat's face. Source: Šimon Klán.

When you come to the virtual environment, you often find yourselve in the presence of other avatars. To maintain a sense of absorption (social absorption) it is mainly important that

1. head movements,
2. moves around the virtual scene,

were natural, although you don't need legs to move around in a virtual environment. The sense of absorption is disturbed when the bodies of avatars, their hands, feets, and gestures move in a different way than you know from the real world. This knowledge, which suggests how a hand or head should move, without examining it, is called *proprioception*.

Note that although it is possible look different in the virtual environment than in real life, it is difficult to change complete behavior. Whether it includes body language, head movements, expressions, etc. Unconscious movements cannot be hidden.

Meeting face to face is probably the best way to communicate. Meeting in virtual reality is probably the second best way to communicate. In virtual worlds, it is possible to meet completely regardless of distance. You stand in front of an avatar or avatars and conduct interviews just as you meet at school, at a party, in a city, or at a meeting.

Current virtual reality technologies are such that the controllers distinguish individual fingers on the hands, the sensors monitor eye movements, the processing of the camera's image causes the lips to move according to the speaking etc., and give the avatars a sense–absorbing reality.

Avatars in the virtual worlds move many times in the art environment, probably in a certain sense of art, and above all in the environment of the human mind's ability to imagine things that don't really exist. On the other hand, like real people, there is an insurmountable desire for a gossip[1]. Therefore avatars are primarily social virtual beings. A social cooperation is a key to the interest and further content of virtual worlds. Avatars are not enough knowledge where in virtual worlds find a fun animated puppet, a fire sword, a bloodthirsty dinosaur or a Marsian environment. Much more important is knowing which avatar endures or hates another avatar, who of the avatars likes, which avatar is honest, who cheats or which is friendly.

Experience has shown that when avatars meet, for example, in an educational or gaming or fantasy world, they do not talk about the elements or experiences of these worlds all the time. Sometimes. But more often, they chatter about other avatars that had a fight, disappointment, or gossip. Gossiping usually turns to offenses or injustices. After all, they are human avatars and it seems that a virtual environment in which there are no life–threatening lions, fast cars or other monsters does not affect this way of communication. Gossiping as well as information where you can find a fun animated puppet will also apply here.

People live in two ways. On the one hand, there is an objective reality of cities, trees and birds, on the other, the imagination of gods, numbers, nations or communities. The imagination becomes stronger with time. Avatars are absorbed by the imagination of the virtual environment. A higher degree of imagination of virtual gods, nationalities or communities is yet to come.

Avatars have the following capabilities in a virtual environment:

a) Transmit information about the immersive virtual world. This facilitates the planning and implementation of joint and complex experience.

b) Provide information on social relationships between avatars. This makes it easier to create larger productive and collaborative groups of avatars based on a telepresentation.

c) Introduce things that do not exist in the real world. This improves collaboration between avatars and innovates their social behavior.

Avatars represent the next powerful creators via collaborative telepresentation[2]. Their social behavior in this early period of virtual environments can be likened to modern virtual hunter gatherers. Virtual experience hunters and virtual knowledge gatherers. They work together, sharing experience, games and knowledge regardless of where they actually physically are.

[1]It is known that human language has not evolved from the need to share information about nature. The most important information to be communicated was about people and not about lions or fish.

[2]A group of people in different parts of the world smoothly interacting as if they were really together.

Their games and experience include coordinated behavior. The capabilities listed above allow them to work together efficiently and in greater numbers, for example, when streaming to Twitch tv platform. Their way of life represents an extraordinary intensive journey to knowledge and skills, and strongly strengthens creative ways of thinking, which in many ways are more intense and effective than previous ways of education. Discovery of virtual species working together in truly large numbers of virtual communities or nations and a faith of virtual gods is yet to come.

The purpose of the avatar groups in virtual environment of Neos metaverse is primarily to share. Games, experiences, and knowledge gain enough time to stay in diverse groups of avatars, free thought exchange, and synergistic decision making. Avatars freely share skills and efforts in cooperating in the provision of experiences. They also share knowledge and virtual merchandise, ie the objects or worlds they formed. This allows them to spend a long time in virtual environment.

The truth is that avatars – individually or in social groups – shape virtual environment in such a wide way that the avatars of the future will likely be surprising. Indeed, when they visit virtual worlds created by avatars long before them, for example, as virtual tourists, they discover that the idea of a historical, non–imaginative virtual landscape they are exploring is an illusion. The avatars before them were related to dramatic changes in the concept of virtual environment, from the densest living jungle to the densest virtual wilderness.

3.2 Appearance, initialization and reincarnation of avatars

The appearance of Avatars typically fits into one of the following groups:

a) Sci–fi caricatures: an anime character, a fairy tale, a fantasy movie etc., taken or originally designed. They are illustrated by avatars in Fig. 3.3. They can be characterized as a lizard type (animal head, two or four limbs, and a long tail), as can be seen from several recordings of live streaming on Twitch in the references (Neos VR Live Stream).



Figure 3.3: Sci–fi caricatures of avatars.

b) 3D scan: real characters scanned by a 3D scanner or simplified human characters. It results in 3D models that can be animated on Mixamo server, as will be seen below.

c) Shape and material: symbolic figures such as photographs stuck on a sphere (see Fig. 2.63), flying head and hands etc. When creating, you can use creative power of the metaverse.

d) Rational animal or robot: creatures with a human body and an animal head as in Fig. 3.2, clever foxes (Fig. 3.3 in the middle). The body is a human and the head an animal. Historically, this is probably the oldest ability of the human mind to imagine such an object and something that actually does not exist. A preserved example of this ability is a Lion Man (or woman) from a cave in Germany, about 32 thousand years old. The ivory Fig. is shown in 3.4.

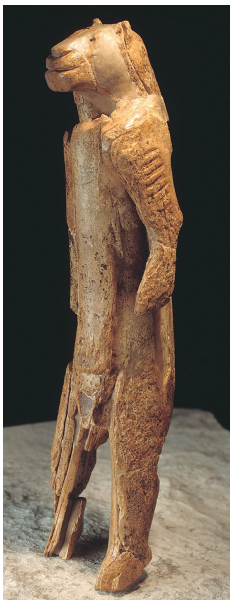


Figure 3.4: The Lion Man (woman) is the first indisputable example of art and probably also of human faith and the ability to imagine things that do not exist in the real world.

e) Mythological figure: characters known from ancient cultures having, for example, the ancient character of the Creator of Fig. 1.16 etc. Finally, the concept of avatar itself has the mythological origin.

Note that the appearance of avatar should not produce a negative emotional response. In this sense, aesthetics uses the term *uncanny valley*, where a computer–generated character or humanoid robot resembling an almost human being, evokes negative emotions of restlessness or even resistance. It is wise to avoid such an effect.

If you have a 3D avatar model created and want to actively use the model in the Neos metaverse (this is how the others will see you), it is time to start the avatar initialization, ie to perform the so–called avatar setup.

The first time you sign up to metaverse, or use the metaverse as an anonymous[3], the multiverse uses its default avatar. The appearance of the default avatar is imitated by virtual glasses as shown in Fig. 3.5. Your username is shown above the glasses. If you nod or turn your head, the avatar moves as well. Hands are presented by palms with fingers. They begin at the wrist and have a computer–generated humanoid character of the palm with differentiated fingers.

[3]If logged into Steam and aren't signed in to the multiverse.



Figure 3.5: Appearance of the default avatar (without hands).

How looks your current avatar you find out easily. Grab the inspector tool and use your thumb to push the controller menu button. Select Create new... and confirm it by one click with your index finger. The menu appears as shown in Fig. 2.56. Use the beam to select the Object item and press the index finger once to confirm. Select Mirror from the next menu and press the index finger to confirm. Then a mirror appears in the virtual world, which can be located and sized as needed. When you look in the mirror, you will see the appearance of your current avatar. The sequence of above actions is represented by the Create new... → Object → Mirror schema.

How to set up a new avatar? You import the avatar in the way you import any other model. From the dashboard, select Tools → File Browser, choose the appropriate file (for example .obj), click on the file to select it and double–click to start the loading process. If the avatar is loaded correctly in the virtual world, you will initialize it.

Grab the inspector tool again, press the small knob of the controller menu with your thumb and select Create New... → Object → Avatar Create. Alternatively, you can also use the Tools from the dashboard menu and with your index finger click directly on Avatar Create. In the world, a triangular ball–head assembly with built–in glasses appears, with symmetrically placed symbolic hands underneath (see Fig. 3.6). This arrangement represents an empty avatar to which you place the prepared avatar.

The inside of an imaginary triangle is filled by a form with buttons. Grab the avatar model (assuming the robotic sci–fi head from Fig. 3.7 left) and insert its head into the head of an empty avatar so that the eyes of the built–in glasses are visible, see Fig. 3.7 in the middle. Form buttons can also be used to align your avatar. If the avatar has only a head, you can leave its hands. In the case that the avatar model will also have hands, put the hands of empty avatar on the hands of your new avatar. To complete the avatar incarnation process, click the Create button. At that point, the entire empty avatar assembly will disappear, leaving only the initialized avatar model. However, looking in the mirror, the active avatar has not changed.

You can only change an avatar if you are logged into the Neos metaverse. You save the model of the initialized avatar to the inventory. You already know the process of storing in inventory. Press the side button to grab the initialized avatar and use the second controller to click + at the top right of the inventory bar. This will save the avatar. If you select the

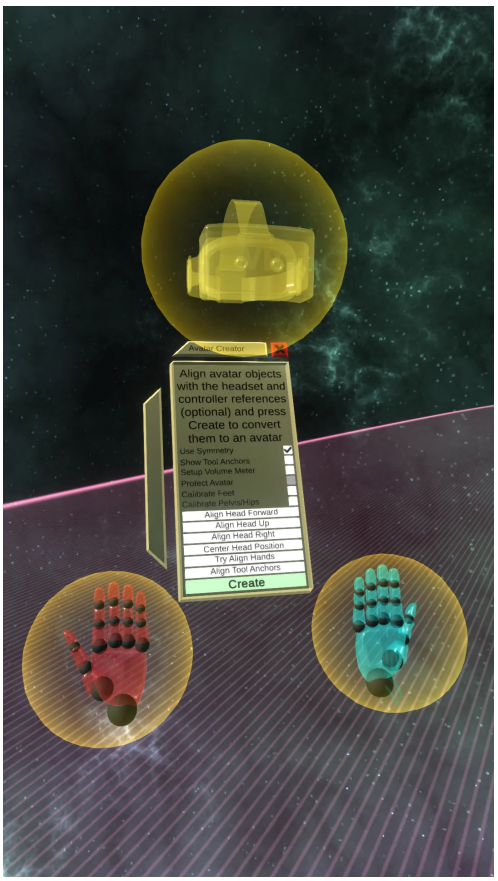


Figure 3.6: Blank avatar set.

saved avatar and click the index finger to confirm, you will see the ↯ symbol in the top bar to the left of +. Just clicking the lightning symbol with your index finger will reincarnate the avatar into a new form. In the mirror you will see the appearance of the new avatar in Fig. 3.7 right, unlike the default avatar in Fig. 3.5. Of course, you can also use a pure human avatar as shown in 3.8.

The avatar is initialized as described above. This makes it possible to change the appearance in the virtual world, ie to reincarnate from one avatar to another, very easily. Select the initialized avatar in the inventory and click the lightning symbol ↯ with your index finger. To set this avatar as the default avatar, click the ♡ heart symbol in the top bar.

Given the importance, let us recall the basic steps to reincarnate the avatar:

1. Initialize with Avatar Create.
2. Save to inventory.
3. Select and click ↯.
4. Click ♡ for the permanent default avatar.

To easily change the avatar, note that the behavior of the avatar can not be changed so easily.

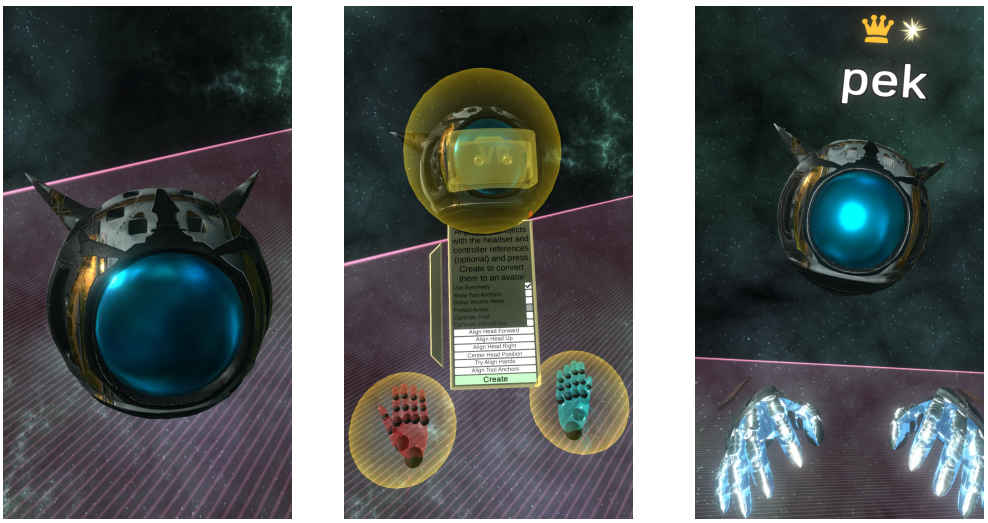


Figure 3.7: Initialization of avatar. Left: Avatar. In the middle: Insert into an empty avatar. Right: Incarnate avatar.

3.3 How to scan an avatar

If the avatar is to represent the same human being in the virtual environment as it is in reality, you will scan the real being. For example, the Structure Sensor of 1.11 can be used, scanning a bust of a human takes about 2 minutes, the entire scan roughly double.

Place the scanned object in a well lit place with spread hands at about 45° from the vertical and with spread fingers. The legs are slightly apart. Scanning always starts with the face. The scanning procedure is similar to the inanimate objects in the `https://www.youtube.com/watch?v=KP1_RVGXh1U` video demo.

If you do not have a scanner, you can use the previously mentioned photogrammetry method to create an avatar. All you need is a camera on your smartphone or tablet and a specialized software. Photogrammetry (also SFM – Structure From Motion) is a method for calculating the location of a point in 3D space from photos captured in a multi–directional way. If you take a set of photos of a person from various directions and upload them into specialized software, the program creates a 3D model.

It is sensible to take something between 60 – 80 photos to capture every little detail. When taking pictures, it is recommended:

1. Move around a human (and any other) object in a circle.
2. Take overlapping photos from about 60 – 80%.
3. Use diffuse light.

Let's repeat that an example of the procedure for photogrammetry can be seen at `https://www.youtube.com/watch?v=IB-PCb5RMOE`.



Figure 3.8: Human avatar.

3.4 Movements and animations of avatars

The unnatural movements of the avatars disturb the sense of absorption. When you scan or create an avatar, it is nothing more than a mass without bones. Bones are important for the movement of avatars. On the way to a naturally moving avatar, it is therefore desirable to take the next step. To implant bones to the avatar. This process is called *rigging*. Yet it depends very much on what you will require from the avatar locomotives. From head movements to fingers or toes. It should therefore be clear where the avatar will have chin, wrist, elbows, knees, groin etc.

Basic rigging of avatars can be done remotely on Mixamo server `https://www.mixamo.com`, where the avatar model will be uploaded after free registration and login. When you view the avatar, you can rig it by placing the marked rings (pairs of rings automatically retain the symmetry) on the chin, wrist, elbows etc. The ringing situation is shown in Fig. 3.9.

Applying Auto–Rigger will rig the avatar according to the placed rings. The rigged avatar is downloaded and imported into the virtual environment using the object import. Then the avatar is ready for initialization and incarnation.

You can also animate the avatar on Mixamo. For example, an avatar might run a synchronized dance like the avatar in Fig. 3.10. You download and import the animated avatar into a virtual environment just like any other object. In the case of a scan, use the Tools → File Browser → 3D SCAN.

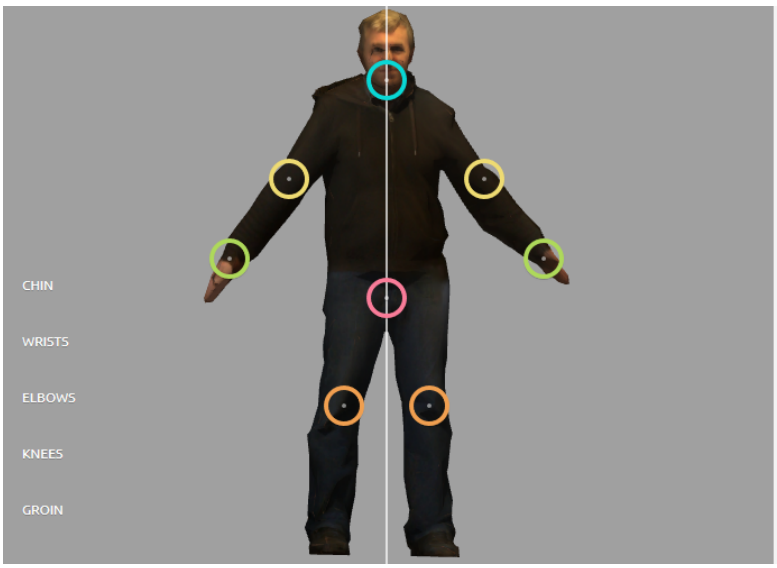


Figure 3.9: Rigging of avatar.

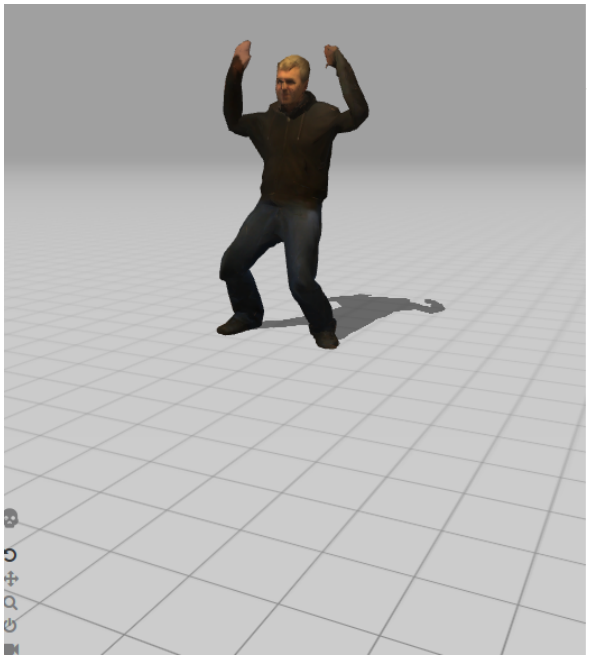


Figure 3.10: Animated avatar.

Example 3.1 *Avatar Shrek.* Import a 3D model of Shrek (see Fig. 3.11) from the animated movie of the same name and realize the reincarnation into this avatar.

Download any of the free 3D models. If the Shrek character is complete, apply rigging and possibly animation in Mixamo. Then you import the model into the virtual world and then gradually initialize it, save it to inventory, and reincarnate. In the virtual world you become Shrek.

Another improvement of avatars are the so–called dynamic bones. Dynamic bones add specific movement (wobbling) to details such as

- hairs,



Figure 3.11: 3D model of Shrek. Source: `https://sketchfab.com`

- items of clothing (for example, a coat),
- ears,
- tail,
- bracelets,
- necklaces.

The principle of working with dynamic bones is that you find the appropriate component in the inspector by Attach Component and add the selected component with dynamic bones to the parent object. Go to item Physics, then to Dynamic Bones and select DynamicBoneChain and click Setup From Children button below. If you then move the object, you should observe the reaction (typically a wave) of its parts, which can be adjusted using stroke potentiometers such as Stiffness, Damping, etc.

The principle of dynamic bones can be easily tested. Using the basic shape tooltip from Essential Tools inventory, you create a chain of three independent cubes. Open the scene inspector, select the second cube, hold the side button and move it to the first cube. This makes the first cube the parent of the child - the second cube. Similarly, you move the third cube to the second so that the second cube becomes the parent of the third. Open the first parent cube inspector and click Attach Component. Select Physics → Dynamic Bones → DynamicBonesChain and click Setup From Children below. Now, for example, when you cut the middle cube with a hand like a sword, you should observe a movement reminiscent of a situation where the cube is interconnected by a spring. The motion parameters can be adjusted using pull potentiometers. An example of such interconnected cubes is shown in Fig. 3.12.

It can also be seen that the linked object cannot be grabbed. So create an empty object (see Empty Object in Fig. 2.56) and copy the linked object into it (mark the left part of the inspector, hold the side button and drag to the empty object, where you release the side button). Add a snap component to the new object via Attach Component → Transform → Interaction → Grabbable and remove those components from each cube. Now, when you grab the first cube and wave it, the other two will flex behind it as in Fig. 3.12. In this way, for example, a flexible snake can be assembled from a series of cubes.

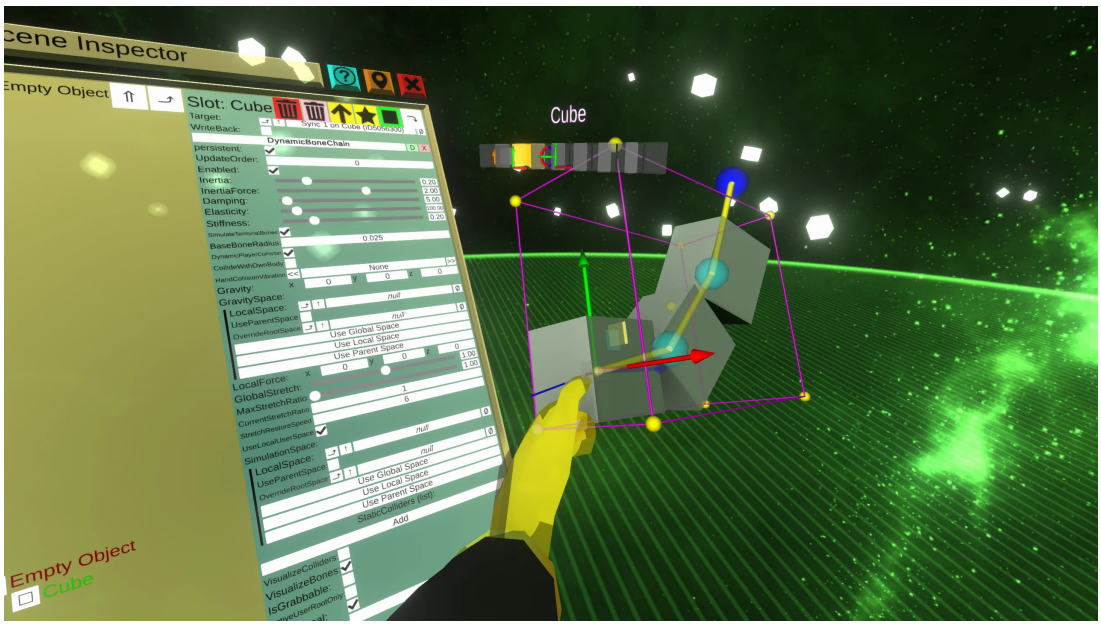


Figure 3.12: Illustration of dynamic bones.

At the end of the avatars chapter, let's note in general that controllers, headset accessories, and other features allow avatars to become more perfected in terms of face tracking, eye tracking, hand, elbow, waist, knee and foot movements for the nature of walking, movement etc. Their use in a virtual environment is manifested, for example, in dance, when the movements of the avatar's body and the expression of his face can be absorbingly synchronized with reality.

Chapter 4

VISUAL PROGRAMMING



Figure 4.1: Abstraction of visual programming of virtual scenes and orbs. Source: unknown author

Each virtual scene object or module can be interactive, change color, size, move, count, respond to impulses from other objects etc. Neos includes the LogiX visual programming system for scene programming. The following important parts will be focused on:

- Mathematical operations with numbers, vectors and strings, use of time.
- Connection with scene inspector.
- Creation of logic schemes, individual color materials and movement of objects.
- Interaction of scene objects, particle systems.
- Performing complex scenes.

Visual programming at first glance increases the complexity of virtual scenes. On the other hand, it can substantially enhance their attractiveness and absorption effect. Fig. 4.1 illustrates the variability of visual scene programming and increasing the complexity of orb content. At the same time, simple strokes say that the motto is to keep the visual program as simple as possible.

4.1 Visual Coding Principle

Creative thinking is usually based on images. Much of human inspiration is happening in pictures. A. Einstein stated that words play no role in his thinking, as opposed to more or less clear images that can be freely reproduced and combined. While writing text helps to clarify thoughts, preparing drawings and diagrams helps to feed ideas. Drawing thus becomes an integral part of the process of creating ideas. It can take form of real images (see 2.17), images, graphs, icons, diagrams, block diagrams (see 4.2), flowcharts, tables, etc.

Basis of the LogiX system is drawing block diagrams such as in the 4.2. The block represents an ingenious invention of engineers according to the idea of keeping it simple. It is a delimitation of certain area of interest into simple block and specification of the quantities that affect the block from the surrounding environment (called inputs) and the quantities that the block itself affects in the surrounding environment (called outputs). A typical block is shown in 1.9. For example, the input from the controllers acts on the virtual environment, and the output from the virtual environment can be, for example, an audio signal. To simplify working with blocks, you typically select one input and one output that are most important for a given block. Blocks, however, have generally more inputs and outputs.

An example of the block diagram is given for the production of electricity as shown in Fig. 4.2. Electricity production can be seen as a system consisting of three principal blocks: water heater, turbine to create mechanical rotational force, and electric generator to produce voltage. The input of the boiler block is given by gas that heats the water in the boiler. The output of the boiler and the input of the turbine block is realized by steam, the pressure of which turns the turbine. The output of the turbine block represents mechanical rotational force, which also enters the generator block. The output of the generator block results in electrical voltage. This shows how block diagrams can simplify situation and plainly illustrates such a complex system as production of electricity is.

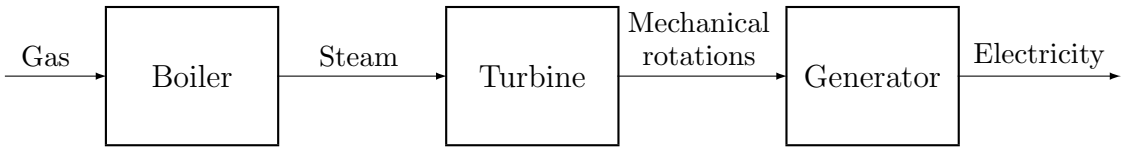


Figure 4.2: Block diagram of electricity production.

Principle of visual programming applied in LogiX is used in Xcos environment for example in numerical computations and visualizations of Scilab[1]. Visual programming or drawing block diagrams consists in simply dragging and dropping blocks from a specific inventory. If the diagram is painted, then the calculation automatically starts.

The basic building elements of visual programming are represented by blocks as shown in Fig. 4.3. Many blocks have single input and single output. Blocks represent mathematical or logical operations, displays, scene objects etc. Visual programming involves selection of blocks from an inventory and connection of them to achieve the desired goal. Connecting blocks can be thought of as a LEGO kit, where the models consist of basic blocks.

Some blocks contain only output. They are called generators. On the other hand, the display elements contain only input.

[1]It is an open–source alternative to well–known commercial MATLAB (MATrix LABoratory) program.

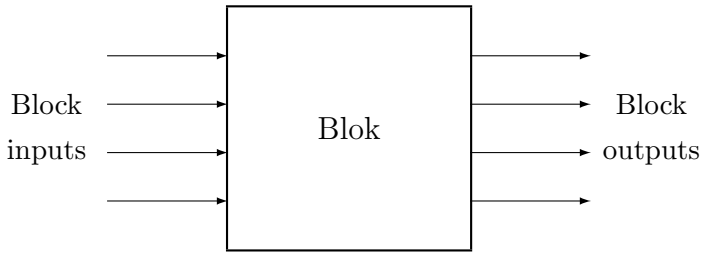


Figure 4.3: Basic block with inputs and outputs.

The principle of working with blocks in the Xcos environment will be evident from the following illustrative example of visualization of the harmonic signal of Fig. 4.4. You can do the same for the LogiX system. From the inventory, select a block named $GENSIN_f$, which generates a sine signal (output). The $CSCOPE$ block is a two–input display that displays the signal in a Cartesian graph. On the horizontal axis the simulation time and on the vertical axis the signal value are plotted. Time is introduced from the clock block. On the display after assembly you get a sine wave graph.

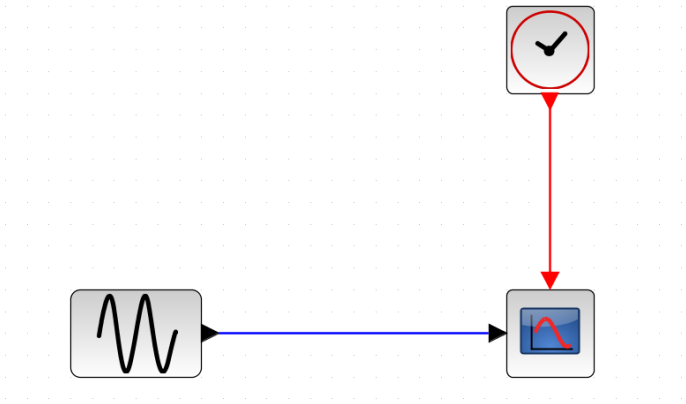


Figure 4.4: Block diagram of sine wave visualization.

4.2 LogiX Tool

LogiX has own executive tooltip. In the Essential Tools inventory, it is a blue tool with word Display (see fig. 4.5). Select the tool with a controller beam, confirm with a single click of the index finger and double–click physically place it next to the tool folder. As you know, you could now grab the tooltip by pressing the controller side button twice.

Grab the LogiX tooltip, press the knob of controller menu with your thumb, select Node Selector, and confirm it with a single click with your index finger. The main LogiX inventory (LogiX Nodes) will appear as shown in Fig. 4.5.

The inventory includes folders with input generating blocks (Input), display blocks (Display), blocks for mathematical functions (Math), blocks for mathematical operations (Operators),

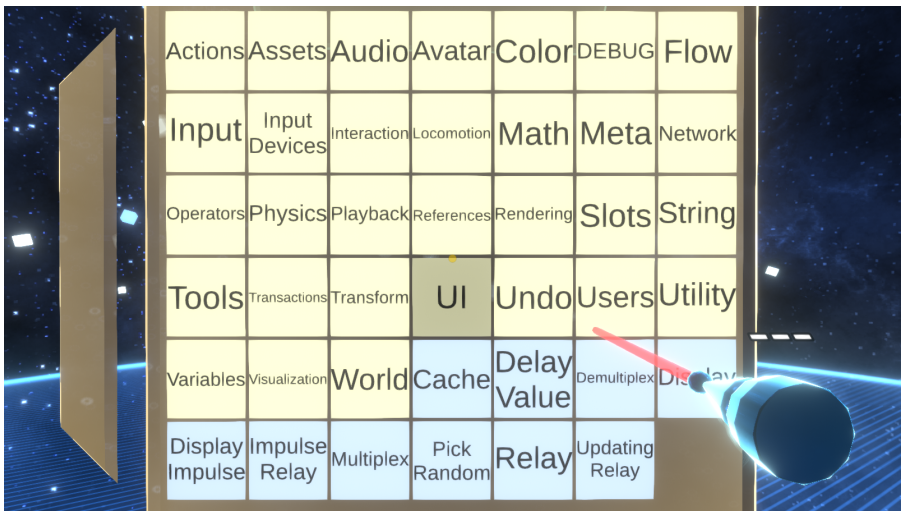


Figure 4.5: Inventory of LogiX tools.

and many others.

4.3 Counting with numbers, vectors and strings

Let's start by simply adding $3 + 7 = 10$. In terms of block diagrams, $+$ can be considered as a block with two inputs and one output as shown in Fig. 4.6. The 3 and 7 numbers are single–output blocks (generators) and the resulting 10 single–input block (display). In the LogiX system you proceed similarly.

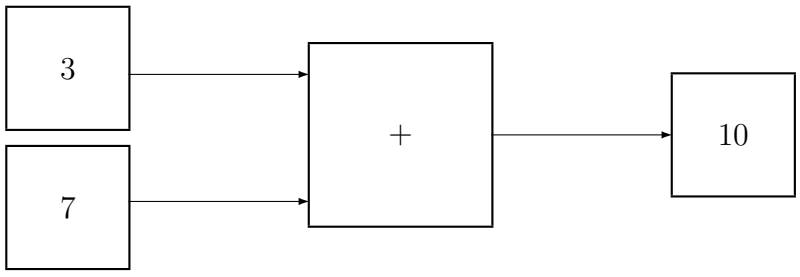


Figure 4.6: Numerical operation $3 + 7 = 10$ block diagram.

Take the Logix tooltip and open its main inventory as shown in Fig. 4.5. Math operations are collected in the Operators inventory folder. Double–click the index finger to open Operators and select the $+$ by the controller beam. Double–click the index finger to charge LogiX tooltip. To observe the charge, the selected operation appears as a label directly above the LogiX tooltip. If you focus on a place in the 3D environment and double–click with your index finger, a sum block appears in this place.

The 3 and 7 numbers represent the inputs. By double–clicking on the back left top of the Operators folder, you will return to the main LogiX inventory. Select and double–click to open the Input folder. Select the float block. Double–click to charge the tooltip, focus on place somewhere to the left of the sum block, and double–click to generate a block for the float number. Since you have two inputs, repeat the double click of the index finger for the second input number.

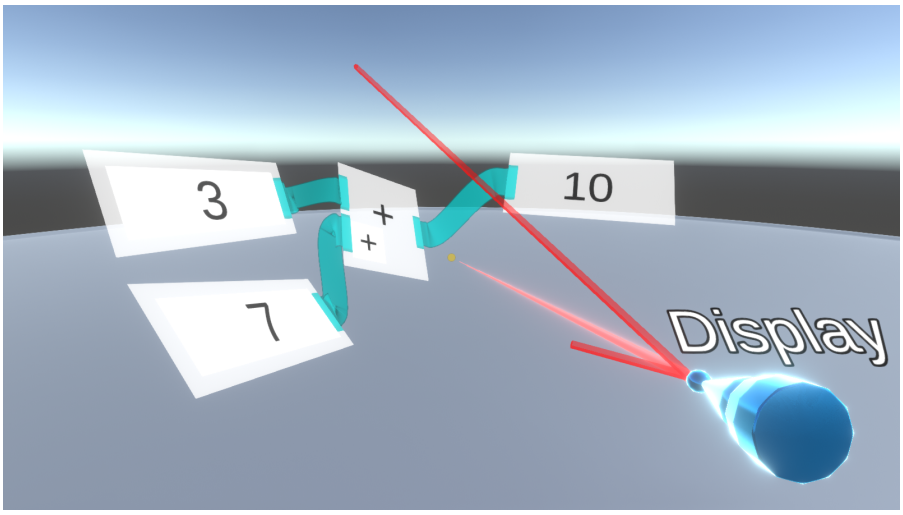


Figure 4.7: The sum of two numbers.

To enter the concrete number into the block, focus the controller beam on the block and click with the index finger. The keyboard appears automatically. You focus the beam on key 3 and click with index finger to enter it. Click Enter to exit. Repeat for 7. In case you make a mistake, you use the ← key to move backwards. You could use other keyboard buttons as usual.

As coded by color on the + operator block, the operator has two inputs[2] and one output. Connect the output of the 3 block to the first sum operator input now. Dive the ball on the LogiX tooltip into the output, press the index finger and pull the controller toward the input on the + operator. At the same time, you could observe that along with the tooltip you maintain a link from the output of the 3 block. If you dip the ball on the controller tooltip into the other end of the connection – the operator input + – and release the index finger, the connection between the output and the input will be established. The connection color is different for different data types:

- blue – float (decimal number),
- green – integer (whole number),
- black – boolean (logic value 0 or 1),
- red – string (chain of characters),
- yellow – color (RGBA color).

If you connect different data types (for example, integer and float), the conversion between them is done either simply by changing the color or by automatic insertion of a conversion

[2]Additional input can be added by focusing the beam on and clicking + in the lower left corner.

block.

In the same way you connect the output of the 7 block to the second input of the + operator. The next step is to display the 10 result. You return to the base LogiX inventory as shown in Fig. 4.5 and select the Display block. Double–click to charge LogiX tooltip, and when you focus on the place to the right of the + operator and double–click with your index finger, a display block appears with one input which is color–coded. Connect the output of the + operator to the input of the displaying unit, just as you have connected the inputs 3 and 7.

Once the link between the + operator output and the display input has been established, the resulting 10 value will appear in the display as shown in Fig. 4.7. Now when you change one of the input values or add another input, the calculation will be automatically updated.

To illustrate the work with vectors, realize a scalar (dot) and a vector (cross) product. The cross product block is found in Operators similar to the sum block in Fig. 4.7. In the same way, you select the block with the controller beam together with the click and double–click to charge the LogiX tooltip. After focusing the controller beam on the selected virtual place and double–clicking, the product block appears in the virtual environment. The input vectors for this operation can be found in the Input folder. Three–dimensional vectors can be found here as float3 types. With a beam and a click select it, double–click to charge the LogiX tooltip, and after double–click move it to virtual space. Since you have two inputs, focus the tooltip below and repeat the double click for the second input. The situation is illustrated in Fig. 4.8.

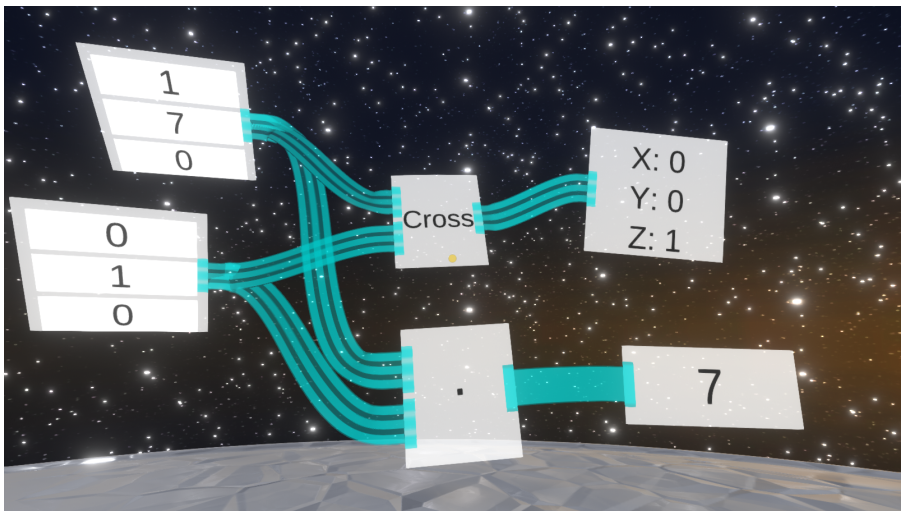


Figure 4.8: Scalar and vector product.

Each vector coordinate is separately entered. Focus the beam on the appropriate field and click. A keyboard appears in the virtual space where you type the digits. Enter exits. The input blocks are connected with the cross product vector block in the same way as in the case of the sum above. Dive the tooltip ball into the output of the vector block, press the index finger and drag toward the color–coded Cross operation input. At the same time you could observe that with the tooltip you maintain the connection from the output to the input vector block. The patch bar is triple to indicate that you are working with 3D vectors. If you dipp the ball on the tooltip into the other end of the connection – the input of the Cross operator – and release the index finger, the connection between the output and the input is established.

In the same way you connect the output of the second vector to the second input of the Cross block. The next step is to display the resulting vector. Return to the basic LogiX inventory as shown in 4.5 and select the Display block. Double–click to charge the LogiX tooltip, and when you focus the beam on the place to the right of the Cross operator and double–click with the index finger, a display block appears with one color–coded input. Similarly to the inputs, you connect the output of the Cross operator to the input of the display unit.

Once the connection between the output of the Cross operator and the input of the display unit has been established, the result of the vector product, i.e. a vector perpendicular to the input vectors, will appear in the display unit. Now when you change any of the input vectors, the calculation is automatically updated.

In the case of dot product $\cdot$, you proceed similarly. Select the $\cdot$ block from the Operators folder and move the block to virtual space. You connect the input vectors (multiple inputs can be powered from one output) and take the Display block from the basic LogiX inventory again. After establishing the Display block input to the output of the $\cdot$ block, the display will show the result as shown in 4.8.

Let's summarize how LogiX inventory blocks appear in virtual space and how the visual programming block diagrams are created:

1. Use the controller beam to select the appropriate block in the inventory and click the index finger to choose the block.

2. Double click to charge the LogiX tooltip.

3. Focus the controller beam on the virtual place where you plan to place the block.

4. Double click to display the charged block at the specified location.

5. Dive the ball of the tooltip into the colored block input/output, press the index finger and drag controller to make connections.

6. Drag the controller beam to the opposite input/output and release the index finger to establish the connection.

As an example of work with LogiX, let's mention concatenation (sum) of strings because it illustrates working with different data types of variables than numbers or vectors. The block diagram creation process remains unchanged. Other illustrative examples can be found in the LogiX Examples world, which is in the hub of the virtual cathedral.

Example 4.1 *Concatenation od strings.* Concatenate various strings – parts of the alphabet – into one long string.

Assume four strings

- abcdefg,
- hijklmn,
- opqrst,
- uvwxyz

and concatenate them into a single string of the full alphabet and count the number of characters in that string. From the Operators folder, use the same + operator as above and transfer the block to virtual space. Focus the controller beam on and click + sign in the lower left of the block to expand the number of inputs to 4. From the Operators folder, return to the basic LogiX inventory and open the Input folder, select the string block and transfer the block to virtual space nad make another three copies. Enter the respective strings in the individual input boxes. From the Input folder, go back to the basic inventory, select the Display block, and transfer somewhere near the output of the + block. Then connect the blocks. Connections are red because working with String data types.

Immediately after the connection you will see the result in the Display block – the whole alphabet similar to the Fig. 4.9. To count the characters of the string, go to the String folder of the basic inventory. Here you select the String Length block, transfer it to virtual space and connect its input to the output of the + block. You already could know that you can connect more inputs to one output. You return to the basic inventory and transfer again the Display block, whose input is then connected to the output of the String Length block. You see 27 characters right away. You can also indicate the green color of the connection color because it represents the integer.

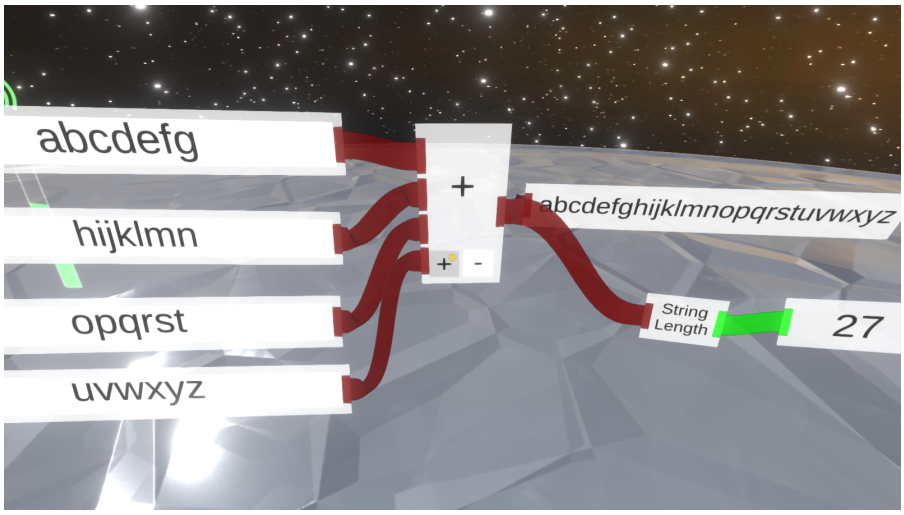


Figure 4.9: Concatenating four substrings creates an alphabet.

4.4 Working with time

The time and date blocks can be accessed from the basic LogiX inventory as shown in Fig. 4.5 by first opening the Math folder and then its Date Time subfolder.

The T block time generator provides much wider time management capabilities. The block can be found in the Input folder. When you transfer and connect this block to the Display block, you discover the principle of its operation. It is a continuous second time counter with the accuracy of approximately tenths of milliseconds, as shown in Fig. 4.10.

For example, you can use the T block to design counter of seconds when you use a rounding block (e.g. floor) to integers. To discover other possibilities of LogiX, such as use of variables, you could proceed in a different way. The counter of seconds can also be provided by

T mod 1 function, where the mod operation denotes remainder after dividing by 1. For example, 47.664 mod 1 = 0.664. Therefore, the values of this function will range between 0 and 1 and the function will be periodic. The operation block mod denoted by % can be found in the Operators folder. The result of the operation can be displayed when using the Display block from the main LogiX inventory.

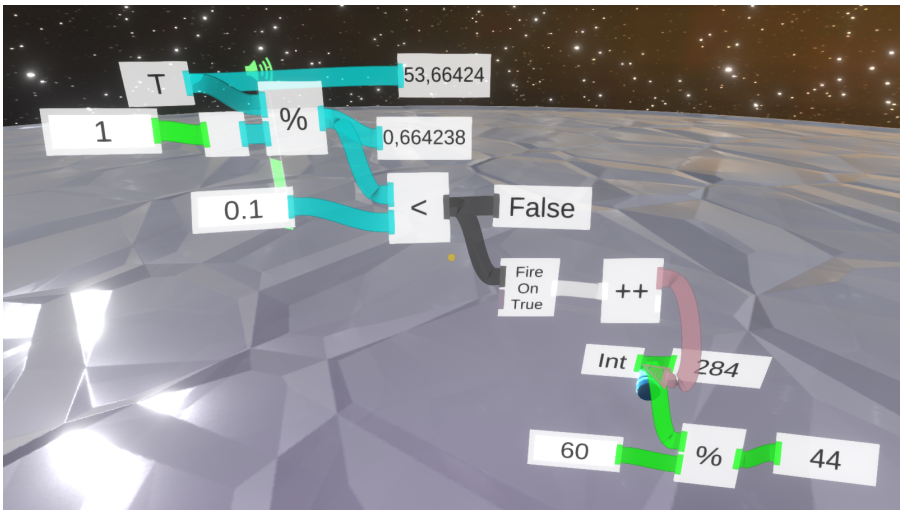


Figure 4.10: Time derived from T variable.

If you test when the value of the function is less than 0.1 for example, you receive a one–second clock. The comparison block is found in the Operators folder, marked with <. The block has two inputs and one output. The first input is the current value and the second the value to which it is compared. The output generates a logic variable, as can be seen in the display block of the 4.10. When you synchronize on true values, you get the counter of seconds. Counters are frequently used, so explain them in more detail.

First of all, it is needed to capture true states of comparison, ie moments when the comparison results are true. To do this, you use a specific block labeled as Fire on True. It fires in the case of the true value. You find this block in the Flow folder of the main inventory. At each firing, the counter value will increase by 1. An increase of 1 is performed by a block marked as ++ which is collected in the Actions folder of the main inventory. Every firing, that is, each impulse will start a 1 increment as shown in 4.10.

The value of the counter of seconds must be stored somewhere. This is done by using variables from the Variables folder of the main inventory. In our case it is an integer, so you select the Int block. Variable blocks do not have classic inputs. When you need to perform an operation on variables, in our case an increment of 1, you do so by outputting the operation to the variable output. In Fig. 4.10 you could see the connection between the output of the block ++ and the output Int. Violet color is used for these types of connections.

If you want to know the current value of the Int variable, you could use the usual universal block Display. If you go further and construct a counter of minutes, you use mod 60 and proceed in the same way as the counter of seconds.

Repeat the key thing again. If you use a block for variables (not only integers and increments of 1, the variables can represent floating point numbers, vectors, etc.), remember that these are blocks that have only outputs. When you require to perform an operation on variables, such as increment or write a new value, you connect output from the increment

block (or any other appropriate block) to the variable block output.

4.5 Color interaction

In visual programming of scenes, a use of logical operations is often required. In the Operators folder of the LogiX main inventory, you could find blocks for basic logical operations AND denoted as &, OR denoted as |, NAND, and NOR. Working with logical blocks is the same as with blocks for numeric operations above. Recall that the connections between logic inputs/outputs have black color. Fig. 4.11 shows the use of a block for the NAND operation. If you use the same logic inputs, you reduce the NAND operation to a simple negation.

In the virtual space you introduce the NAND block. Go to the main inventory folder Input and select the input block labeled bool. Introduce this block somewhere to the left of the NAND block. If you focus the beam on the input value of the block and click with the index finger, you change the logical value of the input. From True to False and vice versa from False to True. The output of the input block is connected to both inputs of the NAND block. You return to the main inventory and select the Display block to be introduced to the right of the NAND block. Then connect the output of the NAND block to the input of the Display block. As shown in Figure 4.11, the negation of the True logic value is False.

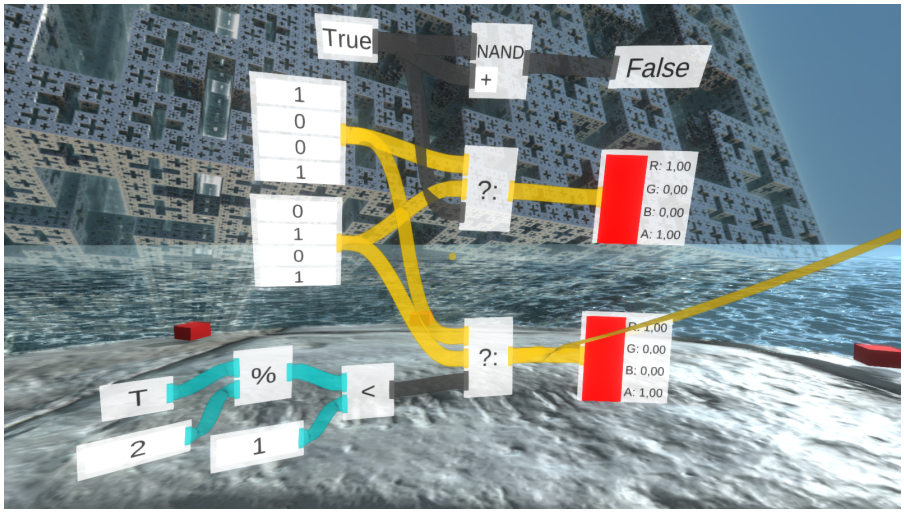


Figure 4.11: A switching of the color range.

Change colors depending on logical variables. Suppose that the value of log. 1 will cause red color and log. 0 green color. In order to experiment, choose a widely used block marked with question mark together with colon ? :. The block can be found in the Operators folder. It has three inputs:

1. Data.
2. Data.
3. Logic control.

This block works so that when TRUE is present on the control input, the first data input is transmitted to the output, and the second data input is transmitted when FALSE. Introduce the ? : Block in the virtual space and connect its control input to the logical input as shown in Fig. 4.11. From the Operators folder, return to the main inventory and open the Input folder. Here you find a block called Color (RGBA) as input. It's a color generator. Introduce the block to the left of the ? : block twice, since you will switch between the two colors. Set the first Color block to red $(1, 0, 0, 1)$ (to set the base color, a number from 0 to 1 range is used) and the second to green $(0, 1, 0, 1)$. Then you connect the output of the first block Color to the first data input of the block ? : and the output of the second block to the second data input similarly to the Fig. 4.11. Next to the ? : output, you could introduce the universal Display block[3] and connect its input to the output of the ? : block. The Display block automatically adjusts to display the color.

If the control input is supplied with log. 1 as in Fig. 4.11 from the Bool block, the Display block will be colored red. When you change the logical value to False, the display turns green. This is done manually. There is therefore a question of how to change colors automatically in a time period.

Suppose you require that the colors red/green change automatically every second: red to green and back to green again. For this purpose, you use the ? : block again, bringing the red and green colors to the data inputs, similar to Fig. 4.11. For the control input, you provide an one–second clock so that there is log. 1 on the input for one second and log. 0 for another second etc. The second clock cycle can be ensured by using the continual T block shown in the previous, performing $T \mod 2$, and comparing it to 1 as shown in Fig. 4.11. If you add a display block to the ? :, the red/green colors will switch automatically every second.

4.6 Object programming

At the moment you can count on numbers or vectors in LogiX, work with time and change colors periodically. Since there mostly is a lot of objects in virtual scenes, it is natural to ask how to apply the acquired skills directly to the objects. For example, how to change the color of an object, how to move an object over time, etc.

Metaverse Neos is basically a modular machine in which you can work with every object in the scene. This feature also applies to visual programming. As you could remember, the metaverzum contains the important component called inspector. The latter gives access to each scene object and allows the properties of the scene objects to be changed in the LogiX system.

Fig. 4.11 provides a visual scheme for switching the green/red color. Create a simple object in the virtual scene now, such as a cube, and require the object to periodically change the color as determined by the output of the above block ? :. The situation is illustrated in Fig. 4.13. So first you grab the basic shapes tool from Fig. 2.12 and produce a cube.

Then you grab a tool for the scene inspector and open the inspector window. In the left area of the list of scene objects you find the corresponding item – Cube. Open the latter with a double click and in the right area you will see individual modules of this object and their properties. You find out that it is the right object by displaying gizmo around the object. Cube colors refer to module PBS_Metallic. Specifically its AlbedoColor property (see

[3]The display block will automatically adapt to the type of connected input. If you connect a number, it will display a number, if you bring a color, it will display a color.

Fig. 4.12).

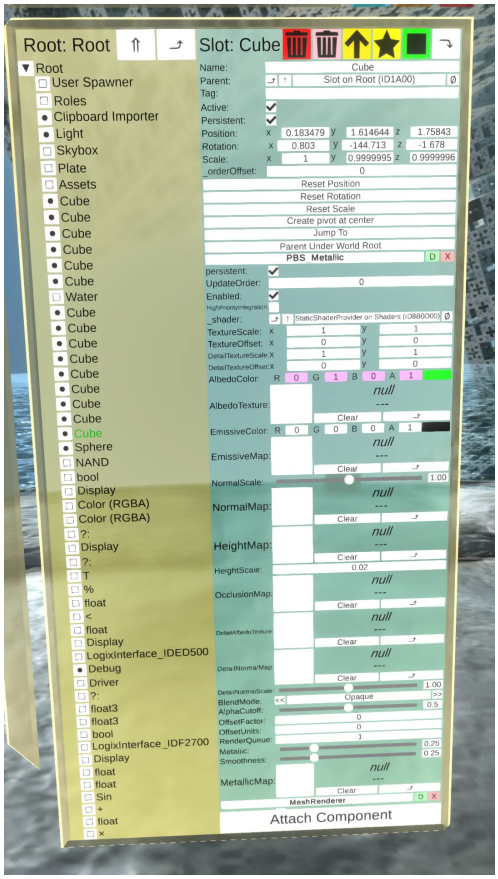


Figure 4.12: The scene inspector window associated with Fig. 4.13.

To make the PBS_Metallic module and its properties usable for LogiX, you could transform it from the inspector into a virtual scene in the form of an input/output block. To transform an inspector module (or the entire object), grab the LogiX tool. Focus the controller beam on the module name, in our case PBS_Metallic, and press the index finger. The module name appears as the LogiX tool label. Drag the module, select an appropriate location in the virtual scene and press the big secondary action knob with your thumb. This will introduce the module blocks to the left of the cube as shown in Fig. 4.13. The module block will also be visually connected to the object to which it relates. In our case, to the cube. Then release the index finger.

This gives you the option to link the output of the above block ? : carrying the current color to the input of the AlbedoColor block, as shown in Fig. 4.13. The cube starts to change color from red to green at one second intervals and vice versa. This completes the process of transforming the module from the inspector and introducing it for LogiX visual programming. In this way, all module features become available for visual programming.

Because transforming either whole objects or their individual modules from the scene inspector window into input/output blocks for use in LogiX visual programming is one of the key moments of visual programming, it follows a brief repeat of the process:

1. Open the scene inspector window.
2. Select the object to be visually programmed.

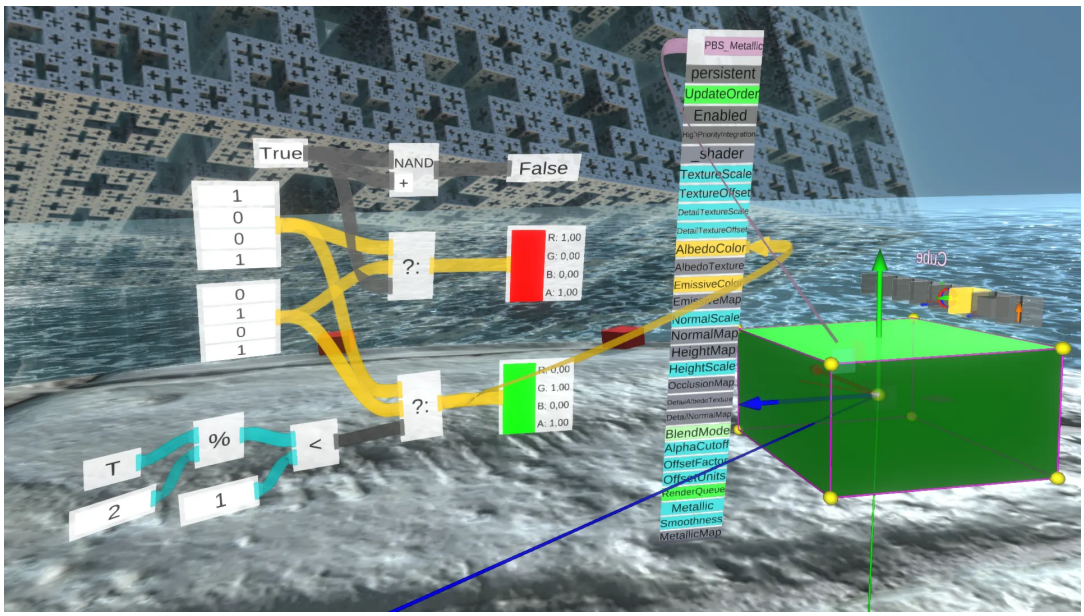


Figure 4.13: Change the color of the scene object.

3. Grab the LogiX tool.

4. Focus either on the name of the entire object or on the name of the module by the controller beam.

5. Press the index finger and drag the module to a suitable location in the virtual scene.

6. Press the big secondary action button.

This introduce the inspector object or individual module in the form of an input/output column block. You use the latter in the same way as the other blocks of LogiX visual programming.

Move from changing colors to moving of scene objects. Using a suitable material, create a stone ball and move it along vertical axis y from one boundary position to another when the logical value changes. The visual programming situation is illustrated in Fig. 4.14. You could apply the known block ? :, where you use data inputs to enter the start and end coordinates of the stone sphere. When you create a sphere in virtual space, the initial coordinates x, y, z can be determined by using the inspector (the Position property). Start with $(2, 1, 0)$ and move the sphere 2.6 m up in the y direction as shown in Fig. 4.14.

As a control input to the ? : block, use the output of bool block from the Input folder. When log. 1, a stone ball will be located in starting position $(2, 1, 0)$ and at log. 0 will move to 2.6 m higher. To prevent the move from happening step by step, you could include a damping block Smooth Lerp from the Math folder of the LogiX principal inventory. The Smooth Lerp block has two inputs – the one you ask to damp and the damping time constant and one output – the one that is damped. Therefore, you connect the undamped output of the block ? : to the first input of the Smooth Lerp block and to the second you connect the output of a float block from the Input folder as the damping time constant.

Open the scene inspector window and find the stone ball object. You take the LogiX tool and focus the beam on the object. Press the index finger and drag the object to the right of the Smooth Lerp block. Then press the large secondary action knob. This will display

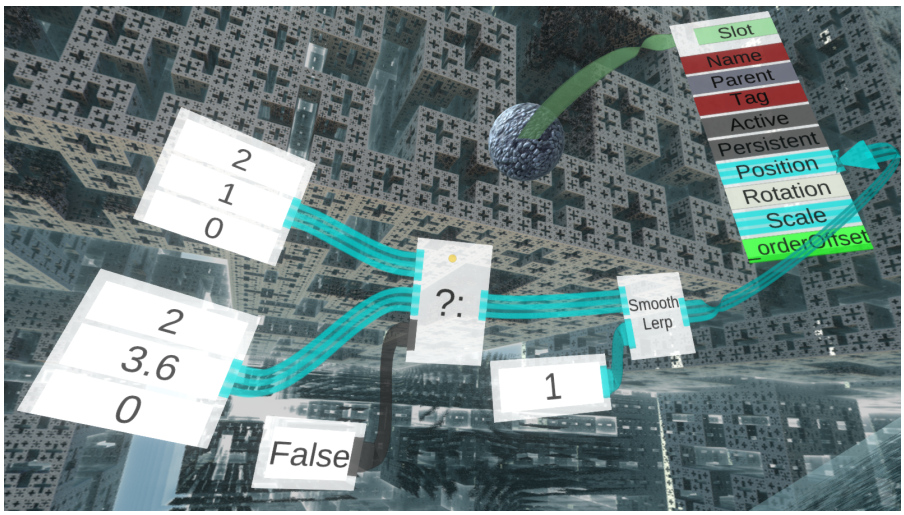


Figure 4.14: Vertical movement of stone ball.

the object in the form of an input/output block column, as shown in Fig. 4.14 right. In the column you could see the Position block representing the position of the object in the form of three coordinates x, y, z. Connect the output of the Smooth Lerp block to the Position block.

Now, if you change the logical value of False to True or vice versa, you will see the ball move from one position to another in a damped motion.

Change the vertical motion of a sphere dependent on logical condition to a periodic motion now. Establish a harmonic movement around the equilibrium height of 4.6 m using coordinates such as

$$\begin{aligned} x &= 0 \\ y &= 3\sin(t/2) + 4.6 \\ z &= 0 \end{aligned}$$

Program the mathematical function $3\sin(t/2)+4.6$ first. From the Input folder you introduce a block labeled $T/2$, which represents time. From the Math folder, you introduce the Sin block and connect them. You need to multiply the result by three. Thus, from the Operators folder, you introduce the $\times$ multiplication block, and from the Input folder, the float block, where you enter 3. Connect the float block output to the first input of the multiplication block and the second input to the output of the Sin block. From the Operators folder, take the + sum block and introduce it to the scene. Switch to the Input folder and use the float block, where you enter 4.6. Now you put all three coordinates together. To do so, you use the Pack xyz block that you haven't used yet. For the block, you go back to the Operators folder. Introduce this block to virtual scene. The block has three inputs – coordinates x, y, z and one packed output, see Fig. 4.15.

According to the assignment you connect the middle input (y coordinate) with the output of the + block and from the Input folder you introduce two float blocks, where you could enter the value 0. You connect the first one with the first input (x coordinates) of the Pack xyz block and the second one (z coordinates) with its third input.

As with the previous line motion, you open the scene inspector window and find the object of the sphere. You grab the LogiX tool and focus the beam on the object. Press the index finger and drag the object to the right of the Pack xyz block. Then press the big secondary action button. This will display the sphere object in the form of a column of in-

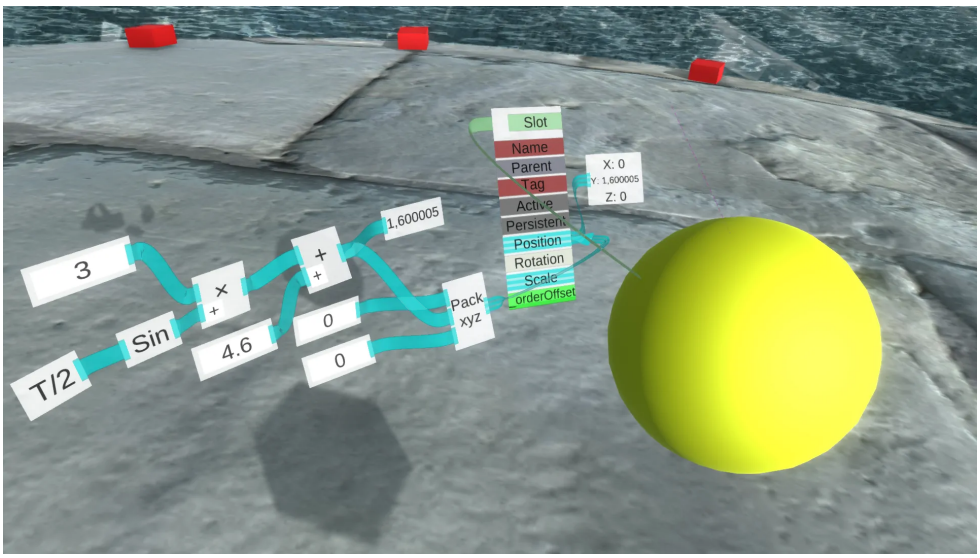


Figure 4.15: Periodic vertical movement.

put/output blocks, as in Fig. 4.15 right. The column shows the Position block representing the position of the object. Connect the output of the Pack xyz block to the Position block now.

Then you can see that the ball starts to move vertically in a harmonic motion. Looking at the exact position, you can use the Display block from the Logix main inventory and connect the block to the Position block. You could observe now, how y coordinate changes periodically over time in accordance with the assignment.

4.7 Complex Scenes

If you examine the main LogiX inventory, you could find that it contains a number of blocks that you haven't used yet. The aim of this chapter is not to describe them all. The aim is to capture a sufficiently advanced methodology for creating progressive content of virtual space. LogiX visual programming is responsible for building complex virtual scenes. Demonstrate therefore two examples of more complex scenes. You will see that your existing knowledge of visual blocks programming is sufficient for this and that knowledge of functions of further and further blocks is desirable only when you need to work with them. In principle, any module or object of a scene can be visually programmed, and the force of this is such that it offers to create various complex scenes.

Pouring beverages

The first example of a more complex scene will be to fill a cup with a drink, tea or coffee. The scene will be a situation where you start to pour a beverage from a teapot and the associated cup starts to fill. When filled, the pouring stops automatically. The whole scene is shown in Fig. 4.16.

First you prepare the basic objects of the scene:

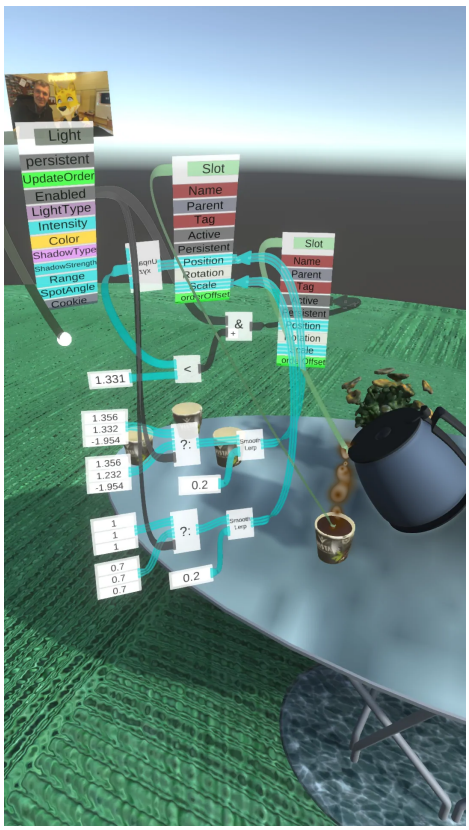


Figure 4.16: Virtual drink serving.

1. Teapot (can) as a source of beverage.
2. Cup (beaker) as recipient of the drink.
3. Particle system as a representative of beverage.

You could scan and import a teapot and a cup into virtual scene. Various available 3D models can be downloaded from the web. You could predefine appropriate scene positions for these objects.

At second, you use the particle system to model a liquid stream. To create an initial particle system, grab the material tool and select Create new... from the controller menu. The initial particle system is introduced to the scene now. Grab the inspector tool and select the Particle System from the scene items in the inspector list. Its gizmo is displayed. You move the particle emitter – the origin of the gizmo coordinates – to the location of the beverage outlet from the teapot (can).

Next step consists in adjusting the particle system parameters in the right part of the inspector window so that the particles create a continuous realistic stream towards the cup – stream, color, particles life etc. The names of the individual parameters of the particle system fit natural properties. If not, the trial and error method will help. The parameters of the particulate beverage pouring system of Fig. 4.16 are shown in Fig. 4.17. The particle life should be set so that their stream does not end somewhere below the bottom of the cup.

In case you have finished with beverage pouring, you could start with creating a beverage surface in the cup. The latter is round and opaque. It indicates that a suitable surface could be a thin slice of a cylinder (similar to a salami slice), the diameter of which would gradually

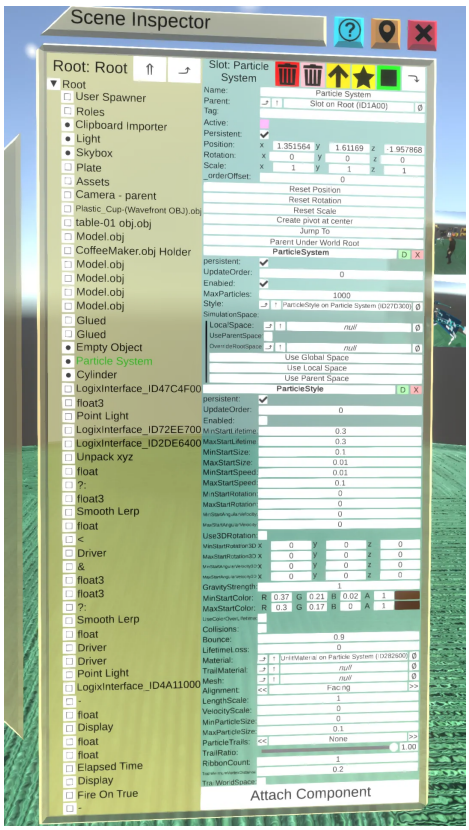


Figure 4.17: Particle system parameters for beverage pouring.

follow the cup from the bottom. Its color simulates the beverage color. Seen from above, a thin slice could provide the illusion of beverage surface well.

You make a thin slice according to the upper edge of the beverage cup and place it just below the upper edge. Related scene simulates the state of the full cup. Using the inspector you find the coordinates of the slice. Then move the thin slice vertically in the y direction to the bottom of the cup. Here, when the beverage is poured, the level begins to rise. Again, you use the inspector to find the coordinates. In the case of the detail in Fig. 4.18, the y axis decreased from the initial value of the full cup 1.332 to 1.232 m, ie 10 cm.

Assume that the pouring is controlled by a logical signal. Log. 1 means pouring and log. 0 stop. At the beginning of the pouring the level will be at the bottom of the cup, at the end it will stop just below the rim of the cup. The bottom diameter will be 0.7 (or 70 % of full scale) of the diameter of the filled cup.

The pouring will continuously increase the surface level. You could use the ? : block from the Operators folder to simulate this phenomenon. When its control input is TRUE, the cup will be full with the surface up. On the other side, FALSE will mean be empty cup with the surface down. When the log. 0 change to log. 1, the cup would fill immediately, which is not possible under physical conditions. But when you use damping, you get an idea that the cup is filled gradually as you know from the real world.

For damping you could use the already known block Smooth Lerp from the folder Math. This block will spread the input jump over time either by linear interpolation or by a limited exponential as shown in Fig. 4.19. If at the time $t = 0$ you switch control input of block ? : from log. 0 to log. 1, the output of the Smooth Lerp block will produce something like a

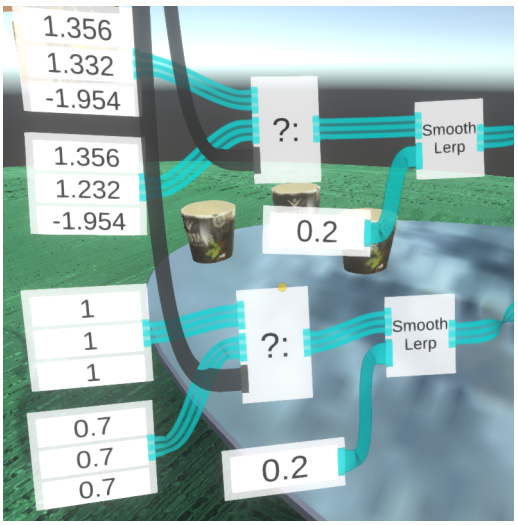


Figure 4.18: Detail of level control.

limited exponential, in our case

$$y(t) = K(1 - e^{-t/\tau}),$$

where constant K determines the gain which is related to jump height (in our case the level of $1.232 - 1.332 = 0.1$) and τ represents a time constant. The larger the latter, the longer the spread will be. The total settling time is approximately given by three times 3τ of this constant. The constant is a parameter of the Smooth Lerp block in the form of $1/\tau$, in our case $1/\tau = 0.2$ (see Fig. 4.18). So $\tau = 5$ and the cup will be filled in approximately 15 s. The slice scale could change in similar way.

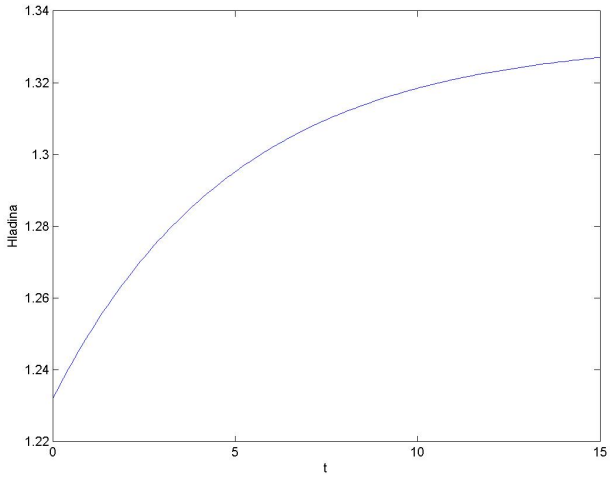


Figure 4.19: Damped level rise.

Open the scene inspector window and select the layer slice object. Grab the LogiX tool and focus on the module name of the object to determine its position and scale. Press the index finger and introduce the module into the virtual scene by pressing the large secondary action knob. Subsequently, you could connect the output of the Smooth Lerp blocks to the position

(item Position) and scale (item Scale) of this module, as shown in Fig. 4.16.

How to solve a logical entry into the ? : block signaling the pouring of a drink occur? To do this, you could use the point light and the corresponding tool from Fig. 2.12 right, which can generate a logic variable. You create light. You could verify that when you click on the light, the light will turn on and vice versa, the next click will cause it to switch off. Open the inspector and introduce the Light module to scene as shown in Fig. 4.20. The Enabled item of this module generates logical signal: log. 1 when light shines and log. 0 when it is off. Connect the Enabled item to the corresponding input of the blocks ? :. When the light comes on, the cup is filled with beverage. Conversely, when you turn light off, it simulates a situation where you drink beverage in the cup.



Figure 4.20: Logic control of pouring detail.

After the light is switched on, you need to start the particle system to show that the beverage is pouring into the cup, causing the surface to rise causally. Reopen the scene inspector, find the particle system, and introduce the particle system module into virtual scene. It contains the Active entry. The particle system will be active when you turn on the light and the surface will be lower than, for example, 1.331 m, ie 1 mm lower than the maximum level.

Take the Unpack xyz block from the Operators folder and expand the surface height. Using the $<$ block from the Operators folder, you could determine if the surface height is less than 1.331 m and at the same time using the AND block whether pouring is on. Connect the AND output to the Active element of the particle system. If this constraint is met, the particle system is active. Once the cup is full, the particle system automatically shuts down because the condition that the level is below 1.331 m is no longer valid.

Now, when you turn on the light, the particle system starts and the cup begins to fill. When you shortly wait, the cup is filled with a beverage to its fulness where the particle system activity automatically shuts down. When the light is turned off, the cup empties without starting the particle system activity. Note that instead of ? : blocks, an integration of small increments can be used to increase the beverage surface.

Powerpoint slide presentation

The second example of a complex scene deals with the virtual powerpoint presentation of slides as shown in Fig. 4.21. First, you could create the date and time when you take the UtcNow block from the Input folder of the LogiX main inventory and connect its output to the input of the Display block. Eventually, it is possible to insert To Local Time block to convert Utc time to your local time before you connect the Display block.

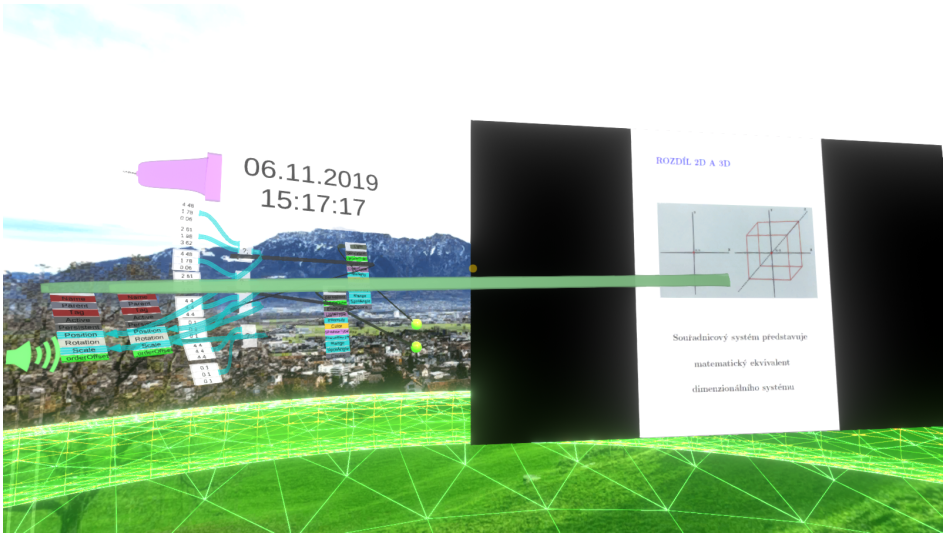


Figure 4.21: Scene arrangement for virtual presentation.

Suppose that the presentation is organized gradually by uploading individual slides of the presentation, for example in JPG or PNG format, and placing them in a collection somewhere in the scene in a way that they are substantially reduced in size in order to be not visible in the scene. The presentation, ie projection of these images on the screen, will be controlled using the buttons. Each slide will have its own button for visual programming simplicity. Pressing the first button will display the first image on the presentation screen, pressing the second button the second, and so on. In addition, it can present several authors collaboratively from different places.

You proceed using blocks ? :. In case of log. 0, the slide will be associated with invisible scaled down state, for example, to 0.1 of its size in a collection set. When the control signal changes to log. 1, the slide moves to the screen and scaled up to 4.4 of its size, similar to Fig. 4.21. Consider to use lights again as buttons.

Assume two slides. Using the tool shown in Fig. 2.12 right, create two associated lights. Open the inspector window and introduce the relevant modules into the virtual scene. Next you will need four blocks ? :, two to change the position and two to change the scale of each slide, similar as illustrated in Fig. 4.22. You open the inspector again and introduce the modules with their positions and scales. For each slide, using the scene inspector, you experimentally determine the starting position and the final position of the screen. You connect these blocks with float3 inputs representing three coordinates x, y and z from the Input folder. For logical inputs you use Enabled elements of both lights. The slides in each axis are scaled up from 0.1 to 4.4 of their size (the z axis is not important in this case because slides are represented by the quad).

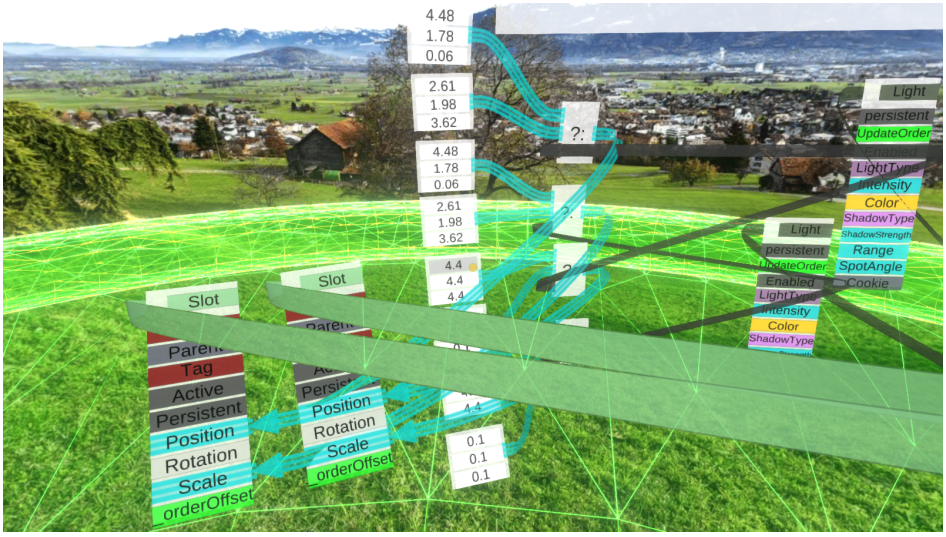


Figure 4.22: Detail of the visual program for the powerpoint presentation.

User blocks

Despite the wide range of LogiX inventory blocks, there are situations where you could require a specific block in scene programming that is not found in the inventory. On the other side, you can create new blocks and use them together with existing blocks. You could apply the specific drawing canvas called Blueprint in Fig. 4.23. This canvas can be found in the Essential Tools → PolyLogiX Public Tools.

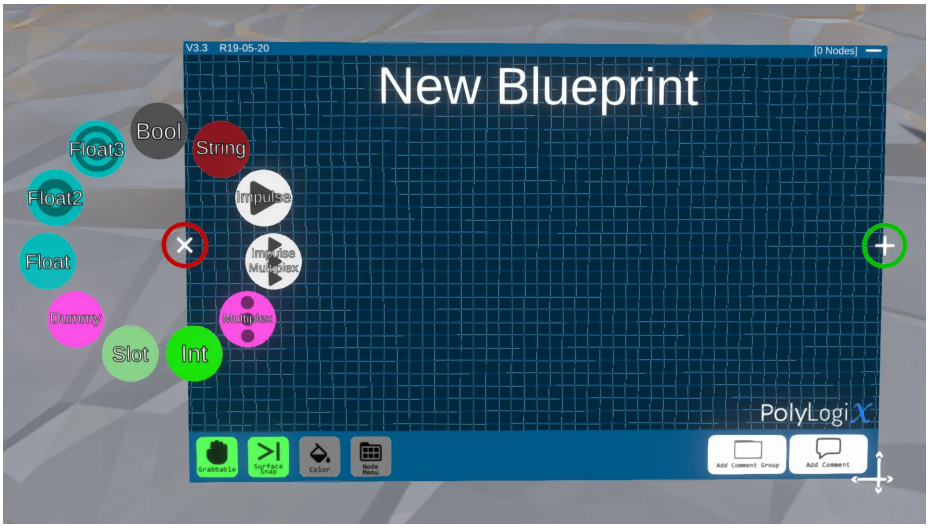


Figure 4.23: Drawing canvas blueprint.

Its principle is a magnetic board, where you clip a block from LogiX inventory, which you connect. On the left and right margins you could see ⊕ symbols, which you click to expand (see Fig. 4.23 left) and select the input or output data types. If you have such a custom block, use the – top right symbol to wrap the block so that it looks similar to other LogiX blocks, ie like a black box with the name and designation of inputs and outputs. The block name is created by overwriting the existing name New Blueprint.

Let's illustrate the whole procedure with an example. Suppose you require a block in the scene making sum of the first n members of the so called harmonic series

$$h(n) = \frac{1}{1} + \frac{1}{2} + \frac{1}{3} + \frac{1}{4} + \ldots + \frac{1}{n}.$$

As you could look at, the harmonic series does not have the finite sum, but its partial sums grow very slowly, see Tab. 4.1.

n	$h(n)$
1	1
10	2.93
100	5.19
1000	7.49
10 000	9.79
100 000	12.09
1 000 000	14.39

Table 4.1: Partial sums of harmonic series.

Open the canvas of Blueprint. Block input could be

1. number of members n you want to add, and
2. logical variable starting the calculation.

The output will determined by partial sums. For example, if you enter $n = 1000$ at the block input, you expect $h(1000) = 7.49$ in be output. You could use two variables in the block. One integer for the order of the members of the series, and the other floating point where the partial sums are stored. The whole scheme is in Fig. 4.24.

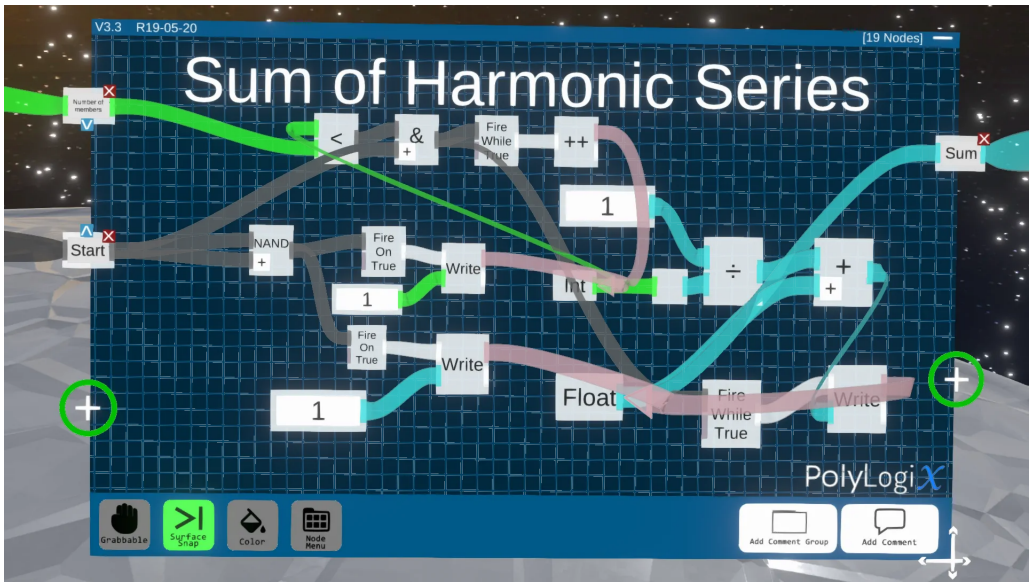


Figure 4.24: Block for partial sums of harmonic series.

Then, when you connect the inputs and display output to the created block, you could verify values from Tab. 4.1, that for example $h(10\,000) = 9.79$.

In conclusion, the controller menu when holding the LogiX tool includes a packaging service so that the visual program is not finally visible on the scene while working fully. Similarly, you can unpack the invisible LogiX schema. The examples in this chapter can be found in the world of Virtual Reality Lectures (VR8 to VR11). Other practical examples for using LogiX in Visual Programming can be found in the LogiX Examples world.

REFERENCES

1. arXiv: virtual reality https://arxiv.org
2. Austin, D. (2008). What is . . . JPEG? *Notices of the American Mathematical Society*, 55, No. 2, pp. 226–229. http://www.ams.org/notices/200802/tx080200226p.pdf
3. DeRose, T. a M. Kass, T. Truong. (1998). Subdivision Surfaces in Character Animation. *Proceedings of the 25th Annual Conference on Computer Graphics and Interactive Techniques*, pp. 85–94. http://graphics.pixar.com/library/Geri/paper.pdf
4. Gray, P. (2013). *Free to LEARN.* Basic Books.
5. Harari, Y.N. (2015). *Sapiens: A Brief History of Humankind.* VINTAGE.
6. Johnsonbauch, R. (1997). *Discrete Mathematics.* Prentice Hall (Fourth Edition).
7. Klán, P. (2017). *Four Components: Modern Theory of Numbers.* TIGRIS (in Czech).
8. Klán, P. (2014). *Numbers: Relationships, Insights and Eternal Inspiration.* Academia (in Czech).
9. Klán, P. a R. Gorez (2011). *Process Control.* FCC Public.
10. Mack, K. a R. Ruud (2019). *Unreal Engine 4 Virtual Reality Projects.* Packt Publishing.
11. McAdams, A. a S. Osher, J. Teran. (2010). Crashing Waves,Awesome Explosions,Turbulent Smoke, and Beyond: Applied Mathematics and Scientific Computing in the Visual Effects Industry. *Notices of the American Mathematical Society*, 57, No. 5, pp. 614–623. https://www.math.ucla.edu/~jteran/papers/MTO10.pdf, http://www.ams.org/notices/201005/rtx100500614p.pdf
12. Neos VR: Wiki, FAQ
    - http://wiki.Neosvr.com/subdom/wiki/index.php?title=Main_Page
    - http://wiki.Neosvr.com/subdom/wiki/index.php?title=Frequently_Asked_Questions
13. Neos VR Basic Avatar Setup https://www.youtube.com/watch?v=N7VKEMTyAl0
14. Neos VR Live Stream https://www.twitch.tv/neosvr
15. Neos VR Full Body IK tweaks & fixes https://www.youtube.com/watch?v=tSFRK_MCPws
16. Pharr, M. a J. Wenzel, G. Humphreys (2018): *Physically Based Rendering: From Theory To Implementation.* http://www.pbr-book.org/
17. Porter, M.E. a J.E. Heppelmann (2017). Why Every Organization Needs an Augmented Reality Strategy. *Harvard Business Review*, November-December Issue. https://hbr.org/2017/11/a-managers-guide-to-augmented-reality
18. Reeves, W.T. (1983). Particle Systems – A Technique for Modeling a Class of Fuzzy Objects. https://cal.cs.umbc.edu/Courses/CS6967-F08/Papers/Reeves-1983-PSA.pdf.
19. Russell, J. (2018). Basic Theory of Physically–Based Rendering. https://marmoset.co/posts/basic-theory-of-physically-based-rendering/
20. Tristem, B. a M. Geig (2016). *Unity Game Development.* SAMS (Second Edition).
21. Virtual Reality in Human Factors Research https://www.youtube.com/watch?v=QWDIid5oh4o&feature=youtu.be
22. Zorin, D. a P. Schrőder. (2000). Subdivision for Modeling and Animation. *SIGGRAPH 2000 Course Notes*, chapter 2. http://www.multires.caltech.edu/pubs/sig00notes.pdf

Notes

Petr Klán, Tomáš Mariančík

How to Build Virtual Worlds in Metaverse Neos

Solirax Ltd

London

2022

ISBN 978-80-11-02566-3