Methods, packages, and applications for the development and use of Mental Cutting Test exercises

Thesis for the Degree of Doctor of Philosophy (PhD)

by Róbert Tóth
Supervisor: Dr. Marianna Zichar

University of Debrecen
Doctoral Council of Natural Sciences and Information Technology
Doctoral School of Informatics
Debrecen, 2023

Hereby I declare that I prepared this thesis within the Doctoral Council of Natural Sciences and Information Technology, Doctoral School of Informatics, University of Debrecen in order to obtain a PhD Degree in Informatics at Debrecen University. The results published in the thesis are not reported in any other PhD theses.

Debrecen, 21/08/2023

.....
Róbert Tóth

Hereby I confirm that Róbert Tóth candidate conducted his studies with my supervision within the Data Science and Visualization Programme of the Doctoral School of Informatics between 2019 and 2023. The independent studies and research work of the candidate significantly contributed to the results published in the thesis. I also declare that the results published in the thesis are not reported in any other theses. I support the acceptance of the thesis.

Debrecen, 21/08/2023

.....
Dr. Marianna Zichar

Methods, packages, and applications for the development and use of Mental Cutting Test exercises

Dissertation submitted in partial fulfilment of the requirements for the doctoral (PhD) degree in Informatics

Written by Róbert Tóth certified Computer Science MSc.

Prepared in the framework of
the Doctoral School of Informatics of University of Debrecen
Data Science and Visualization programme

Supervisor: Dr. Marianna Zichar

The official opponents of the dissertation:

Dr. .....
Dr. .....

The evaluation committee:

chairperson:	Dr. .....
members:	Dr. .....
	Dr. .....
	Dr. .....
	Dr. .....

The date of the dissertation defence: 2023. .....

Contents

1. Introduction

In this dissertation, we present our research to enhance the development of exercises for the Mental Cutting Test (MCT). In this period, we dealt with the design and development of web- and mobile applications which focus on the students and the improvement of the learning process, offering practice exercises with gamified elements and Augmented Reality (AR). However, we realized after the alpha release of our mobile application that generating MCT exercises should be the primary research goal since there is no generally available database or package to retrieve or construct exercises. Moreover, the COVID–19 pandemic made it impossible to connect with students and instructors personally and organize measurements with the application in a controlled environment. Thus, we concentrated on the exercise generation problem and achieved our main scientific results with the script-aided generation of MCT scenarios and the lossless encoding of their GLB assets. Thus, we give an overview of research areas on improving spatial skills, including use cases of Augmented Reality, Virtual Reality (VR), and Gamification. Later we describe the modeling phase of assets in Blender and the automatic permutation and post-processing, and encoding of them. Finally, possible use cases for the methods will be present, such as the alpha version of our gamified Android application and the generally available web applications, as well as our pilot measurement made at our faculty in 2022.

1.1. Background

Measuring and developing spatial skills

Spatial abilities, such as visuospatial perception, spatial visualization and orientation, mental folding, mental cutting, and mental rotation, play an important role in everyday life as well as in solving complex mathematical problems and engineering design tasks (see, e.g., Bohlmann and Benölken, 2020; Bishop, 1980; Tosto et al., 2014; Cole et al., 2021; Zimmermann, Cunningham, and America. Committee on Computers in Mathematics Education, 1991; Raju, S. Sorby, and Reid, 2022; S. A. Sorby, 2009; Llorens, Contero, and Alcañiz, 2020; Bosnyák and Nagy-Kondor, 2008; Ault and John, 2010; Ben-Chaim, Lappan, and Houang, 1988). The development of spatial skills in the teaching

of STEM fields is a challenging task that may require various tools in various pedagogical situations, including planar drawings, real spatial models, and software (Presmeg, 2020; Presmeg, 2008; Gerber, 2020). The primary goal is to map the optimal and effective combination of these tools, including virtual and augmented reality applications, which were proven to be successful in the effective development of spatial skills and competencies (Alpiste Penalba, Torner Ribé, and Brigos Hermida, 2019; Huerta et al., 2019; Estapa and Nadolny, 2015; Chen, 2019; Cerro Velázquez and Morales Méndez, 2021; Petrov and Atanasova, 2020; Flores-Bascuñana, Diago, Villena-Taranilla, and Yáñez, 2020; Suselo, Wünsche, and Luxton-Reilly, 2021). A recent systematic overview of this topic can be found in Papakostas, Troussas, Krouska, and Sgouropoulou, 2021. Overall, there is a significant demand for applying emerging technologies that could support the development and evaluation of spatial skills through electronic versions of classical tests.

Using emerging technologies

Several researchers started to deal with VR in the last decade and design applications aiming to improve the users' spatial skills with the VR function (Rizzo et al., 1998; Lochhead, Hedley, Çöltekin, and Fisher, 2022; Hartman et al., 2006). This approach proved effective and yielded better results than simple paper-and-pencil exercises (Safadel and White, 2020). However, the initial popularity of VR and the motivation of users started to decrease. The main reasons are clear. First, using VR requires investing in special headgear and powerful hardware. This means that students either have to buy their own devices to participate in these exercises or universities and researchers must provide enough devices for their students. Manufacturers have started to develop plastic or paper frames where users can put their mobile phones and imitate the functions of a VR headset. This provides a less expensive but low-quality method of creating the necessary environment. On the other hand, using VR headgear frequently causes discomfort, so many users cannot wear it (or only wear it for a limited time) (Saredakis et al., 2020; Chang, Kim, and Yoo, 2020).

The demand for enhancing the use of spatial tests and the limitations of VR technology can be proved by checking more publications from the last two decades. Researchers accessed the use of VR in various contexts, such as measuring the skills of secondary and university students by comparing the results

of genders (Di and Zheng, 2022; Guzsvinecz, Kovacs, et al., 2018; Guzsvinecz, Sik-Lanyi, Orban-Mihalyko, and Perge, 2021; Carbonell-Carrera and Saorin, 2017; Guzsvinecz, Orbán-Mihálykó, Sik-Lányi, and Perge, 2021; Parsons, 2004). However, it is obvious that most of these researches focus on evaluating skills instead of developing them. This feature can be derived from the strong limitations on the possibilities. On the other hand, we can find papers describing different experiences using AR (Shaghaghian, Burte, Song, and Yan, 2022; Alves et al., 2017; Ali et al., 2017; Tang, Owen, Biocca, and Mou, 2003; Bogomolova et al., 2020; Garzón, Pavón, and Baldiris, 2019; Gün and Atasoy, 2017; Purnama, Andrew, and Galinium, 2014; Gecu-Parmaksiz and Delialioğlu, 2018; Llorens, Martín Gutiérrez, Contero, and Alcañiz, 2020). The number of papers and their published results verify that a cheap, easy-to-access technology offers more opportunities to measure, develop, and train spatial skills.

Using gamification

After starting to offer practice exercises for users, the most important goal is to engage them and prevent them from losing their motivation. Thus, a Spanish research group has already published results about using gamification for spatial training skills (Llorens Rodríguez et al., 2022). Gamification is a widely used principle that joins different disciplines, such as pedagogy, psychology, and technology. It aims to engage and motivate people with elements that are derived from games. Gamification is used in many areas of life, especially in business and education (Troussas, Krouska, and Virvou, 2017; Browne and Anand, 2013; Barata, Gama, Jorge, and Gonçalves, 2013; Maxim, Brunvand, and Decker, 2017; Huynh, Zuo, and Iida, 2018). In the last decade, researchers introduced many different definitions and dealt with analyzing the patterns and effects of gamification (Ponick and Stuckenholz, 2019; Decker and Lawley, 2013; Swacha and Muszyńska, 2016); thus, the implementation strongly depends on the actual interpretation. However, Karl M. Kapp gave an abstract definition for gamification (Kapp, 2012), which can be used for each scenario and contains all the important features:

"Gamification is using game-based mechanics, aesthetics, and game thinking to engage people, motivate action, promote learning, and solve problems."

Focusing on Mental Cutting Test

There are various, now standard tests to measure these skills. One of these tools is the so-called Mental Cutting Test (MCT), which was originally designed as a part of a college entrance examination test (CEEB, 1939). There are also recent adjustments to this test (Quaiser-Pohl, 2003; Cohen and Hegarty, 2012), but the classical test is still among the most frequently applied method to assess spatial skills, and the form of tasks are standard in each version. An MCT task (also called scenario) contains a 2D projection of a 3D shape and a cutting plane. The exercise seems simple: determine the shape created by intersecting the 3D shape with the plane and select the figure representing the solution out of five candidates. However, scenarios can be constructed with different levels of difficulty. Previous research showed that the difficulty of an exercise strongly depends on the shape and similarity of possible answers (Németh, Sörös, and Miklós Hoffmann, 2007; Nagy-Kondor and Esmailnia, 2021). If alternatives contain easy-to-recognize differences, a scenario proves to be easier; however, if several alternatives contain very similar features and differ only in minor details, the exercise becomes more complicated. A vast number of papers studied the outcome of these tests in various contexts (see, e.g., Bölcskei, Gál-Kállay, Kovács, and Sörös, 2012; Šipuš and Cižmešija, 2012 and references therein).

During the last decades, most researchers and educators used a well-known sheet containing 25 scenarios derived from various shapes and cutting planes[1], with widely permuted possible answers resulting in a wide range of difficulty levels (Gorska, Sheryl Sorby, and Leopold, 1998; Nemeth and Hoffmann, 2006). However, this limited number of available scenarios highly restricts the effective practice and improvement of spatial skills in education.

[1] Available: `https://viskillz.inf.unideb.hu/resources/mct-classic.pdf` (accessed: 13th August 2023)

1.2. Our research

Our research was started in 2019. Our initial goal was to develop a gamified mobile application to offer its users MCT exercises. To support them during the practice, we implemented an AR-based viewer with the use of which users are able to check the scenarios of the exercises in 3D without losing information in 2D. Additionally, the implementation of a VR activity was planned to offer another tool for visualization. Moreover, we used the most popular gamified elements to engage and motivate users to use the application. This time, the *viSkillz Play* pilot application was finished.

Using Blender, we created a schema (see Figure 1.1) for constructing exercises from assets. In our procedure, instructors design models and yield assets required to form an exercise: the 2D image of a scenario (containing a mesh and a frame representing a cutting plane) and five intersections served as possible answers. Following the original format of MCT exercises, one shape of the set is the correct answer, and four others are served as invalid, wrong answers. We realized that the easiest way to form an exercise is to derive these additional shapes from other scenarios.

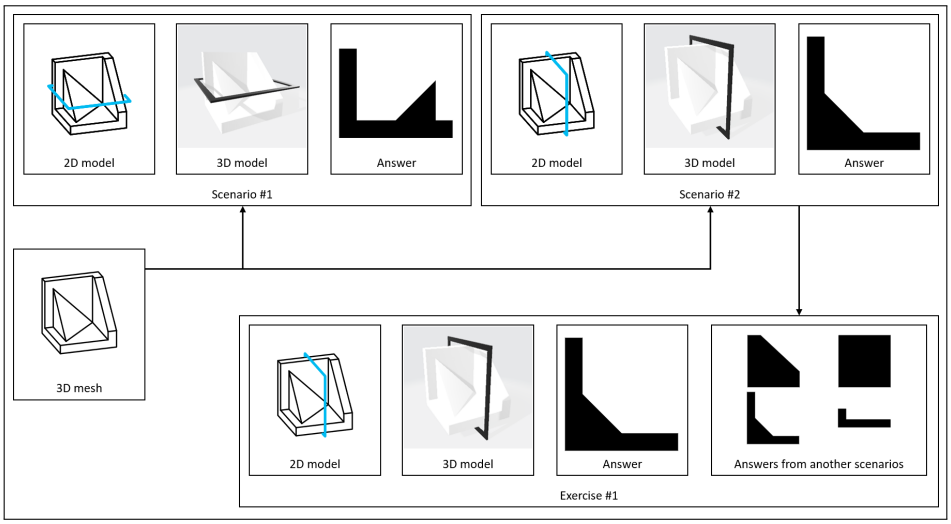


Figure 1.1: Our schema for constructing an MCT exercise from multiple scenarios.

However, we soon realized that the original vision could not be implemented without solving significant issues related to MCT exercises: There was no well-known database offering hundreds or thousands of exercises. There

was no well-known process to generate exercises. There is no proven test creation method based on the users' skills. Moreover, we realized that the manual design of assets requires high effort from instructors, and a more efficient method should be developed to perform the necessary steps.

We started new research directions to enhance the process of developing and using the MCT exercises. First, we designed over 200 models based on the well-known sheet and 19 cutting planes in Blender. Second, we implemented a script-aided method using Blender's Python API, which could automatize most of the required steps to yield assets for exercises. Finally, the first version of our dataset allowed us to examine the assets, including their possible permutation factors, common anomalies, and the limitations of the process.

Parallel, we realized that a mobile application is not the best tool to test students' visual skills since most schools and faculties need to be better equipped with tablets or mobiles. Moreover, bringing one VR headgear to a classroom is not suitable. Parallel, Samsung suspended the support of its Galaxy VR technology, predicting a drop in the popularity of VR technologies. Thus, we started to focus on the technical background of the assessments while moving our developments to the web platform. This time, we found solutions for problems related to miscalculations of Blender and extended our script-aided method to support additional intersection planes and permutation factors. Developing our applications *viSkillz Browser* and *viSkillz Quiz*, we encouraged into another issue: the dataset containing hundred thousands of possible permutations required too much storage space; thus, we were unable to deploy the applications using the entire dataset. In the next phase, we designed a multilevel encoding scheme to store the 3D assets efficiently, enabling developers to manage the dataset. Furthermore, we developed our vector-based matching method to detect similarities between the shapes of intersections. Parallel, we published applications *viSkillz Browser* and *viSkillz Quiz* to test the encoding methods. Finally, we organized a pilot measurement at our faculty to gain experience with the web-based 3D viewer and test whether it affects the results of students having various knowledge and experience about spatial tests and 3D games.

After giving the most important definitions, we will provide a conclusion about the research. The structure of this dissertation follows the logical sequence of our milestones. We must mention that this order differs from their chronological order since we moved the applications to the last chapter. However, the gamified mobile application was essential to realize the need for our procedures related to generating, post-processing, and encoding the MCT scenarios.

1.3. Terminology

In this dissertation, the following important definitions will be used.

Definition 1.1 (Exercise)**.** An MCT exercise contains an image on which a 3D shape and a frame denote a cutting plane with five 2D shapes as possible answers, describing shapes of possible intersections.

Definition 1.2 (Scenario)**.** An MCT scenario is a subset of an MCT exercise without having additional, wrong answers. Thus, a scenario consists of a 3D shape, a cutting plane, and their corresponding intersection.

Definition 1.3 (Assets)**.** The classic assets of a scenario are the initial 2D image which contains the 3D shape and the cutting plane, and a 2D image representing their intersections. Moreover, we use an additional 3D model containing the 3D shape and the cutting plane.

Definition 1.4 (Mesh)**.** We refer to 3D shapes as meshes to make the reader able to distinguish the 3D and 2D shapes without being wordy.

Definition 1.5 (Shape)**.** We only refer to 2D shapes as shapes to make the reader able to distinguish the 3D and 2D shapes without being wordy.

Definition 1.6 (Intersection)**.** An intersection is a shape resulting from the cutting operation performed as part of a scenario.

2. Creating MCT scenarios in Blender

2.1. Introduction

In this section, we give details about our dataset of MCT scenarios and their assets. First, we designed 25 shapes in Blender which are derived from the well-known set of 25 classic exercises. Second, we permuted them manually, yielding 205 different shapes and forming groups. Later we designed and implemented a script-aided method capable of automatically permuting scenarios from these meshes, applying pre-defined cutting planes, rotations, and scaling operators on them.

2.2. Designing scenarios manually

First, we describe the most important features of the Blender project and the method with the use of which we manually design the 3D shapes and other objects representing the cutting planes.

2.2.1. Origin of meshes

Our primary goal was to minimize the difference between the original images and our models, while some features of the shapes had to be handled carefully to avoid rendering and processing anomalies. On the other hand, the figures of the sheet contain easy-to-recognize errors that can be derived from the sequence of continuous printing and scanning operations that resulted in the current state of the paper sheet. For example, some lines are non-parallel should be anyway, and relationships between the models and intersections are sometimes broken. The 2D projections of the shapes can be seen on Figure 2.1.

2.2.2. Structure of the project

We created four collections to contain all the Blender objects that are required to support the automatic generation and rendering of assets as well as make calculations possible in the further steps:

- Collection `Shapes` – The first and most important collection is the collection of 3D shapes that will be intersected (or cutted) with the given

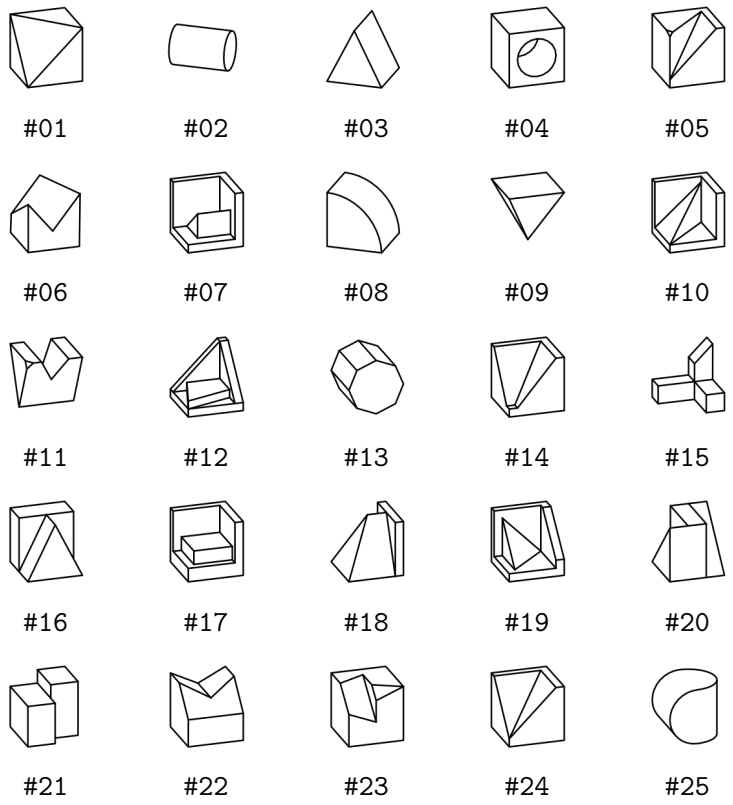


Figure 2.1: Reproduced meshes of the well-known sheet of exercises (`https://viskillz.inf.unideb.hu/resources/mct-classic.pdf`).

cutting plane. The projection of these shapes can be seen in the original 2D exercises. However, the shapes are mostly drawn using white faces and black edges in test exercises. Thus, we added a white material to these objects to support the further generation of 3D assets.

- Collection `Frames2D` – The set of 3D objects that are the frames representing cutting planes in 2D assets. We minimized the surface of these meshes since only a series of segments are required to represent them in the rendered assets.

- Collection `Frames3D` – The set of 3D objects that are the frames representing cutting planes in 3D models used in AR, VR and 3D viewers. These objects have black materials to guarantee a contrast between the 3D shapes and the frames of cutting planes. Moreover, they have sig-

nificant surface and volume, differing from the elements of collection `Frames2D`.

- Collection `Cameras` – Multiple cameras are used to retrieve all the required 2D assets: MCT exercises use two isometric perspectives as well as the shapes of intersections should also be rendered to be served as (possible) answers.

The structure of the project is essential as the base for the script-aided permutation method since it uses reflection to identify and manipulate each object.

2.2.3. Convention for shapes

We had to create a convention for the 3D objects' lengths (sizes) and structure to guarantee that all the generated assets would be similar:

- Each mesh has a maximum length of 2.0 in each dimension[2], while at least one dimension exactly has a length of 2.0. Thus, our meshes have consistent sizes, which makes further processing steps easier and deterministic.
- Origins are also set carefully to the median of each dimension to support rotation transformations.
- In the case of some meshes, we made a few minor adjustments to avoid containing a large number of edges and vertices in the used cutting planes. Although instructors should avoid useless vertices and edges, manually performed subdivisions are not required in most scenarios.

2.2.4. Cleaning meshes

However, manually designed meshes can also contain minor errors that can be resulted from floating precision errors. Most models can be designed easily using Blender's *Boolean* modifier, which computes the intersection, the difference, and the union of two meshes. During the modeling phase, we detected common miscalculations of this modifier:

[2]We used *meter* as the unit in Blender, but any measurement unit can be used, of course.

1. Unnecessary, redundant vertices (and edges) are often added to the output, which changes the most important features of each mesh, resulting in non-deterministic intersections.

2. The created vertices have precision errors that can be derived from the computations performed on floating types.

We will show later that we cannot totally avoid these anomalies, but we can minimize the possibility of their appearance. Thus, we tried to model our meshes without using the *Boolean* modifier and creating most of the vertices, edges, and faces by starting from simple shapes and applying atomic operations manually to them. However, Blender offers a great set of *Clean Up* methods that are worth applying to our models:

- *Delete Loose* – Removes disconnected vertices and edges from the mesh.

- *Degenerate Dissolve* – Removes typically unwanted geometry from the mesh, such as sliver edges, face corners with no area, and empty faces.

- *Make Planar Faces* – Modifies faces that bend beyond the given limit. This operation yields planar faces, while their change also produces non-deterministic differences in the coordinates.

- *Split Non-Planar Faces* – Splits faces that bend beyond the given limit. It subdivides the non-planar faces into planar sub-faces. This operator cannot be used in the design of meshes, while additional edges and slopes will change the main features of the intersections.

Another important operation in Blender's *Edit Mode* is called *Triangulate Faces*. This operator traverses all the faces and splits them into multiple faces to form a mesh containing only triangles. That step has to be involved in the modeling phase to serve as a base for better computations.

Each mesh was checked for non-planar faces using the *Split Non-Planar Faces* operator with a limit of 0.5°. Next, each mesh was modified manually until this operator had no more effect on the mesh, guaranteeing that each face was bent under an acceptable limit. Then each face was triangulated using the operator *Triangulate Faces* with its default parameters. Finally, operators

Degenerate Dissolve and *Delete Loose* were applied on each mesh, in that order, resulting in clean meshes except for outlier cases caused by the use of the *Boolean* modifier.

2.2.5. Designing manual permutations

To yield a large set of permutations in the upcoming steps, we derived additional meshes from the original shapes that belong to the well-known MCT exercises. In this step, we changed one or more main features of the original meshes, resulting in very similar alternates that differ only in symmetry or ratios in most cases. On the other hand, additional features are introduced, or an existing feature is deleted to retrieve a permutation. Usually, we created nine alternates of each original shape (see Figure 2.2). A mesh can be referred to by its unique ID in format `#GGMM`, where `GG` denotes the ID of the group (derived from Figure 2.1) and `MM` denotes the ID of the actual permutation. ID `#GG00` always belongs to the original shape from which the permutations are derived. A set of permutations will be referred to using the `#GG` identifier and called *shape group* or *group*. The contribution of an instructor ends in this step, except for the validation. Thus, all further permutations will be applied deterministic and automatized.

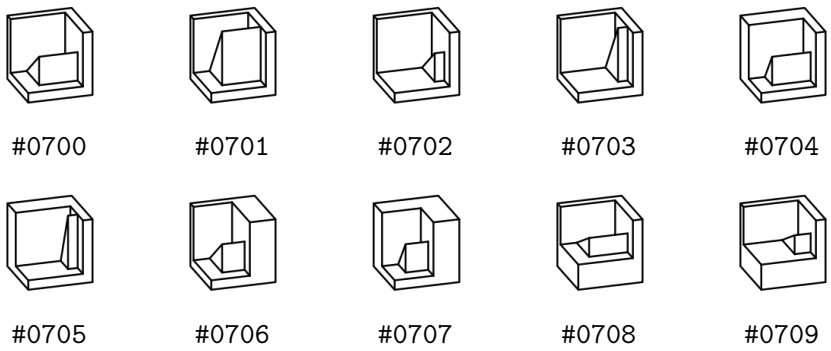


Figure 2.2: We changed the main features of mesh `#07` to obtain new permutations. As a result, most of the cutting planes will yield different intersections from various meshes having the same orientation.

2.2.6. Choosing types of assets

We had to determine which media types would represent our assets. We had to choose one media type for the classic 2D assets and another for 3D models. The following criteria were considered using this process:

- A standard media type should be selected, supported by various technologies.
- We should minimize the size of the assets since a great number of scenarios will be permuted and stored in a database.
- We should be able to serve the assets immediately for HTTP requests sent from mobile or web applications.
- We should find a format for storing data and style separately since all the assets will be styled the same. Moreover, different applications should be able to change the appearance of assets.
- Multiple screen sizes should be supported. Thus, a vector-based type must be selected for 2D assets.

Thus, we started using GLB documents (The Khronos® 3D Formats Working Group, 2021) to represent 3D assets and SVG documents (MozDevNet, n.d.) to represent 2D assets. SVG (*Scalable Vector Graphics*) is an XML-based file format describing two-dimensional vector graphics. Thus, content can be displayed in the same quality in multiple screen sizes, and the data can be stored in text formats. GLB (*GL Transmission Format Binary file*) is a widely used format for 3D assets, represented by JSON-formatted, binary, and image files. However, the specification supports embedding JSON and binary data into a single, standalone file, making the assets easy to handle.

2.2.7. Using *FreeStyle*

We chose Blender's FreeStyle renderer[3] to render our scenes, yielding vector graphic assets using format SVG. The plugin can be manually enabled in Blender's settings and configured using the UI. Moreover, the Python API also

[3] Available: `https://docs.blender.org/manual/en/latest/render/freestyle/introduction.html` (accessed: 13th August 2023)

supports the use of FreeStyle. The plugin can automatically determine which edges should be rendered, but the rendering can be configured to specify rules on edges. Moreover, we can mark edges as *FreeStyle edges* both manually or using the Python API. Thus, multiple `<g>` elements (the container type for groups) can be created in the SVG documents, providing the opportunity of customizing their styles. We defined three groups setting their criteria:

- Group *Shape* – Edges with type *Edge Mark* or *Contour*, using collection `Frames2D` (exclusive).
- Group *Frame* – Edges with type *Edge Mark*, using collection `Frames2D` (inclusive).
- Group *Answer* – Edges with type *Edge Mark* or *External Contour*, using collection `Tmp`[4] (inclusive).

Using these groups, we can create multiple layers in the SVG assets, making post-processing scripts able to identify the edge type automatically.

2.2.8. Cameras

As the first approach, both the 2D projections and the shapes were rendered directly from Blender, using Camera objects. Thus, we had to set their locations and other properties to follow the original MCT features as much as possible. Blender offers five lenses, including the *Perspective* and *Ortographic* ones, but properties of an *Isometric* camera should be set carefully to yield a real isometric perspective, preserving parallel lines. MCT exercises use a special axonometric system with two possible locations of a camera, so we set three cameras to support them as well as the rendering of answers:

- `Camera1` – Used to render the 2D projections of scenarios.
  - location: `(3.3, -1.15, 1.4)` (in meters)
  - rotation: `(68.5, 0, 72)` (in degrees)
  - scale: `(0.923, 0.712, 0.923)`

[4] Temporary, generated objects are assigned to this collection.

- `Camera2` – Used to render the 2D projections of scenarios.
  - location: `(3.3, 1.15, 1.4)` (in meters)
  - rotation: `(68.5, 0, 108)` (in degrees)
  - scale: `(0.923, 0.690, 0.923)`
- `CameraA` – Used to render the shapes of answers.
  - location: `(0, 0, 3.2)` (in meters)
  - rotation: `(0, 0, 90)` (in degrees)
  - scale: `(1.000, 0.772, 1.000)`

2.2.9. Remarks

We realized in the further steps of our research that minor adjustments could be made in this process to yield assets faster and have better quality:

- The use of FreeStyle can be avoided in the case of the intersections if we yield their contour in Blender and export the sequence of segments in JSON format. This output can be processed with a simple Python script, and corresponding SVG documents can be built and serialized using package `minidom`.
- The style of line segments can be more customized post-processing the assets based on the `g` elements representing the layers in the case of 2D projections. Additionally, the assets representing intersections can be created using any style. The possibility of using CSS rules or inline style depends on the platform.
- We use two materials to texture our 3D objects, but Blender adds an unnecessary UV map to each object by default. We have not found any possibility of manually removing this UV map from multiple meshes. Thus, we recommend keeping the UV maps and removing them using scripts.

2.3. Permuting scenarios automatically

Second, we describe our script-aided method, which aims to permute MCT scenarios automatically. The input of this phase is the set of manually designed permutations of 3D shapes and the collection of cutting planes (and their frames that represent the planes).

2.3.1. Applying scaling

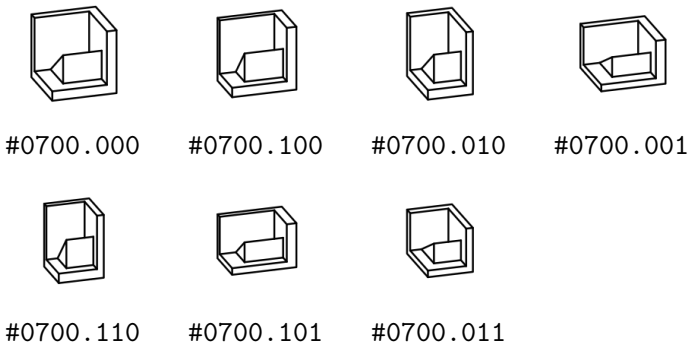


Figure 2.3: The automatically created permutations of mesh #07. Additional shapes can be created by applying each scaling vector S on the same manually permuted mesh.

The features of a mesh – especially the aspect ratio of its edges, thus the aspect ratio of the derived intersections – can be changed by scaling each mesh in one or two global directions; thus, we introduce a set of scaling vectors having values 0.7 and 1.0 in different combinations. Figure 2.3 shows how seven meshes can be derived from mesh #0700 applying scaling vectors $\{1.0, 1.0, 1.0\}$, $\{0.7, 1.0, 1.0\}$, $\{1.0, 0.7, 1.0\}$, $\{1.0, 1.0, 0.7\}$, $\{0.7, 0.7, 1.0\}$, $\{0.7, 1.0, 0.7\}$ and $\{1.0, 0.7, 0.7\}$, where scalar values are given in order XYZ. After introducing the seven scaling vectors, a mesh can be referred to with an ID having format #GGMM.SSS where the subsequence GGMM is inherited from the previous manual permutations, while section SSS denotes the ID of the applied scaling vector of the actual shape in which digit 1 denotes scaling factor 0.7, and 0 denotes scaling factor 1.0. Thus the sequence can be interpreted as a sequence of flags indicating whether the shape is scaled on the actual global axis.

2.3.2. Applying rotation

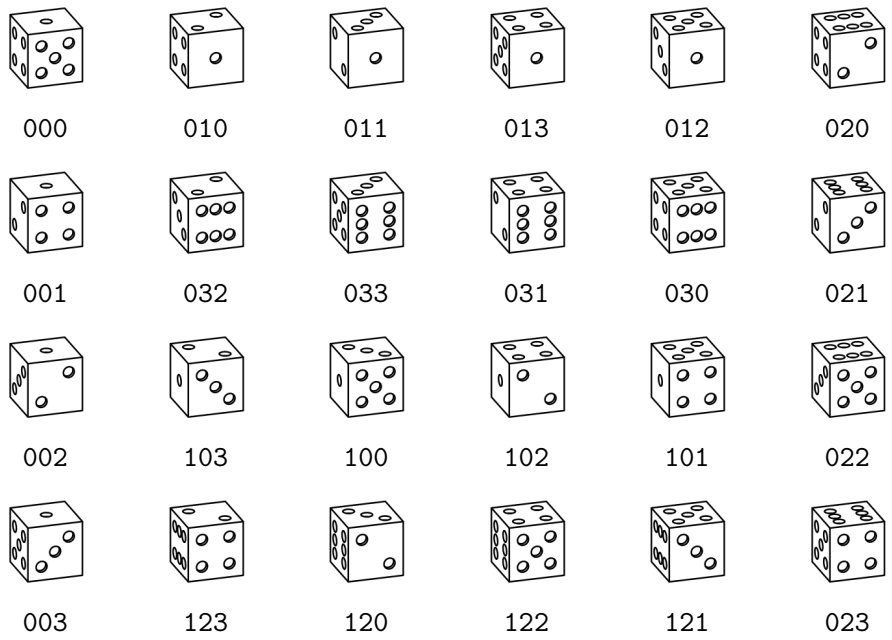


Figure 2.4: Different rotations of a dice. The sequence of numbers under each image denotes the unique ID of the rotation in the specified format. With the use of 24 different rotation vectors, each orientation of the cube can be achieved.

Additional permutations can be easily generated from each mesh using rotation transformations. As we will intersect each mesh with various planes, the actual orientation of the mesh can change the shape of the intersection. Denote the set of rotation vectors as S^{RV} and a single rotation vector as $r = \{X, Y, Z\}$. Each component of any $r \in S^{RV}$ refers to the amount of rotation applied on the corresponding global axis. The amount of rotation is indicated by a value from set $\{0, 1, 2, 3\}$ that is used as the coefficient of 90° when the counter-clockwise rotation along the corresponding axis is performed. For easier notation, we prefer to give literal values in degrees. Thus, each shape orientation is referenced with an ID with the format `#GGMM.SSS.RRR` where subsequence `RRR` denotes an element of S^{RV} (see Figure 2.4).

2.3.3. Defining cutting planes

We already know how the meshes can be permuted; only the set of cutting planes should be chosen. In the initial version of our iterative algorithm, we introduced 19 cutting planes (see Table 2.1) that can be defined by combining a point on them and their normal vector in Blender. In a real exercise-designing process, we must check whether a cutting plane results in a non-empty, easy-to-understand shape. However, this research goal focused only on the permutation, and our goal is to yield as many different shapes as possible. Thus, all possible combinations of shapes, scaling, rotation, and cutting planes should be involved in our computation. Twelve additional cutting planes can be derived from the well-known sheet of paper to yield more intersections, which can result in various intersections. For example, the exercise of shape #09 has a cutting plane that bisects two adjacent sides of the mesh at their center. These planes can be derived from our P10-P15, changing the position of their given coordinates from the origin to coordinates $\{0,-1,0\}$, $\{0,1,0\}$, $\{0,0,-1\}$ and $\{0,0,1\}$, combining these values with the original normal vectors. A total of 12 additional intersections can be yielded using these combinations (see Table 2.2).

2.3.4. Permuting intersections

As a first approach, we can combine each scaling vector with each possible rotation vector, then with each cutting plane. That would yield a total of $7 * 31 * 64 = 13888$ combinations for each manually permuted mesh with much redundancy. On the other hand, we can easily reduce the set of rotation vectors and cutting planes to minimize the redundancy ratio. First, most of the cutting planes can be omitted from the computations. In the meantime, they can be substituted with the combination of another cutting plane and rotation vector:

1. P04 and P07 can be substituted with P01;
2. P03, P05, P06, P08 and P09 can be substituted with P02;
3. P11, P12, P13, P14 and P15 can be substituted with P10;
4. P17, P18 and P19 can be substituted with P16;
5. P21–P32 can be substituted with plane P20.

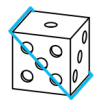
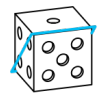
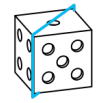
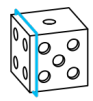
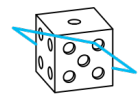
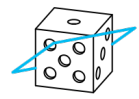
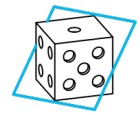
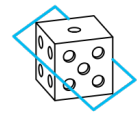
ID	Example	Features	ID	Example	Features
P01		C=(0, 0, 0) N=(0, 0, 1) R=(0, 0, 0)	P10		C=(0, 0, 0) N=(0, 1, -1) R=(-45, 0, 0)
P02		C=(0, 0, -0.8) N=(0, 0, 1) R=(0, 0, 0)	P11		C=(0, 0, 0) N=(-1, 0, -1) R=(0, -45, 0)
P03		C=(0, 0, 0.8) N=(0, 0, 1) R=(0, 0, 0)	P12		C=(0, 0, 0) N=(0, -1, -1) R=(45, 0, 0)
P04		C=(0, 0, 0) N=(0, 1, 0) R=(90, 0, 0)	P13		C=(0, 0, 0) N=(1, 0, -1) R=(0, 45, 0)
P05		C=(0, 0.8, 0) N=(0, 1, 0) R=(90, 0, 0)	P14		C=(0, 0, 0) N=(1, 1, 0) R=(90, 0, 45)
P06		C=(0, -0.8, 0) N=(0, 1, 0) R=(90, 0, 0)	P15		C=(0, 0, 0) N=(1, -1, 0) R=(0, 90, 45)
P07		C=(0, 0, 0) N=(1, 0, 0) R=(0, 90, 0)	P16		C=(0, 0, 0) N=(-0.5, 0.5, 1) R=(0, -35, -135)
P08		C=(-0.8, 0, 0) N=(1, 0, 0) R=(0, 90, 0)	P17		C=(0, 0, 0) N=(-0.5, -0.5, 1) R=(0, -35, -225)
P09		C=(0.8, 0, 0) N=(1, 0, 0) R=(0, 90, 0)	P18		C=(0, 0, 0) N=(0.5, -0.5, 1) R=(0, -35, -315)
			P19		C=(0, 0, 0) N=(0.5, 0.5, 1) R=(0, -35, -45)

Table 2.1: The originally introduced, 19 cutting planes.

ID	Example	Features	ID	Example	Features
P20		C=(0, 1, 0) N=(0, 1, -1) R=(-45, 0, 0)	P26		C=(0, 0, 1) N=(1, 0, -1) R=(0, 45, 0)
P21		C=(0, -1, 0) N=(0, 1, -1) R=(-45, 0, 0)	P27		C=(0, 0, -1) N=(1, 0, -1) R=(0, 45, 0)
P22		C=(0, 0, 1) N=(-1, 0, -1) R=(0, -45, 0)	P28		C=(0, 1, 0) N=(1, 1, 0) R=(90, 0, 45)
P23		C=(0, 0, -1) N=(-1, 0, -1) R=(0, -45, 0)	P29		C=(0, -1, 0) N=(1, 1, 0) R=(90, 0, 45)
P24		C=(0, 1, 0) N=(0, -1, -1) R=(45, 0, 0)	P30		C=(0, 1, 0) N=(1, -1, 0) R=(0, 90, 45)
P25		C=(0, -1, 0) N=(0, -1, -1) R=(45, 0, 0)	P31		C=(0, -1, 0) N=(1, -1, 0) R=(0, 90, 45)

Table 2.2: The secondly introduced, additional 12 cutting planes.

Thus, it is sufficient to use only five cutting planes (`P01`, `P02`, `P10`, `P16`, and `P20`) to fetch all the possible permutations. With the help of this information, we can prove that the 64 rotation vectors can be substituted with only 24 ones. To check it, consider a dice with six unique faces where each one is identified with the number of dots on its surface, while the sum of dots from two opposite faces is seven exactly. Various sequences of 90° rotations can change the orientation of the dice. To count the number of different orientations, we can choose a face that has to be on the top, then consider which face is on the front side of the dice. We can see that a cube can be rotated into $6 * 4 = 24$ unique orientations (see Figure 2.4), which means that the use of 64 rotation vectors can be reduced to only 24 carefully selected rotation vectors. Thus, the following set of rotation vectors will be applied on each mesh to yield all the possible permutations:

1. Each rotation vector in which $R_x = 0$;
2. Each rotation vector in which $R_x = 90$ and $R_y \in \{0, 180\}$.

2.3.5. Computing and exporting intersections

Given a permuted mesh, a scaling vector, a rotation vector, and a cutting plane, the computation strictly follows the corresponding step of our iterative algorithm. If a mesh intersects with a given cutting plane, then the intersection is cleaned and flattened to the `XY` plane. The shape is translated to the origin and scaled finally. In the iterative algorithm, each shape was rendered at this point, resulting in SVG and PNG representations. However, we will process the coordinates of each edge and its vertices. Thus, instead of rendering, we fetch the local coordinates of each edge and transform them into the global coordinate system. Each rotation vector is used in Blender's `XYZ` Euler coordinate system. Algorithm 2.1 shows the steps of rendering SVG and PNG assets, then exporting GLB assets.

Algorithm 2.1 Permuting and exporting assets of a manually designed mesh

```
1: function CREATE-ASSETS(s_o, c)
2:     for each s_s ∈ SCALE-OBJECT(s_o) do
3:         l_o = MOVE-OBJECT(s_s, (0,0,0))
4:         SHOW-OBJECT(s_s)
5:         for each f ∈ F_2 do
6:             SHOW-OBJECT(f)
7:             for each r ∈ R do
8:                 ROTATE-GLOBAL(s_s, r)
9:                 EXPORT-SCENE()
10:            end for
11:            HIDE-OBJECT(f)
12:        end for
13:        HIDE-OBJECT(s_s)
14:        MOVE-OBJECT(s_s, l_o)
15:    end for
16:    CLEAR-COLLECTION(T)
17: end function
```

Each shape is scaled before calculating intersections with the vector $\{20,20,20\}$ to guarantee better precision, resulting in a shape with a maximum length of 40 in each dimension. The `bisect()` operator is applied to this state of the mesh. After the post-processing steps, each shape is scaled with vector $\{10,10,10\}$ to achieve a maximum length of 20 in each dimension. That guarantees that each coordinate comes from the interval $[-10,10]$. Finally, the edges of each intersection are dumped into a JSON document to provide a structured dataset for the upcoming Blender-independent steps of our method.

The 25 original meshes have various features; however, the handling of four ones is not obvious if the geometry is described with edges instead of curves. These meshes are `#02`, `#04`, `#06` and `#25`. These meshes will not be involved in our future computations, as a specific algorithm is needed to handle them correctly. This specific algorithm uses the number of edges to compare the two problematic intersections.

2.4. Conclusion

2.4.1. Results

This chapter introduced our Blender-based workflow for designing and retrieving MCT scenarios using the well-known software's standard UI and Python API. First, we examined the standard sheet of 25 classic MCT exercises and determined the common features of the exercises and the possible permutation factors. This process also included the reproduction of the 25 well-known 3D meshes. Second, we chose the crucial Blender tools required to perform the calculations on the manually designed objects. As a result, we can retrieve both the 2D and 3D assets necessary to form an MCT scenario. Finally, we introduced the extended process for generating assets using 31 pre-defined cutting planes, 7 scaling operators, and all the possible rotations. Instructors can yield hundreds of unique scenarios based on a single manually designed 3D mesh using our approach. More than 1 million scenarios can be retrieved from the 205 manually created meshes using the permutation factors.

2.4.2. Availability

Our Blender project, Blender module, and a standalone Python wrapper can be found in our GitHub repository: `https://github.com/viskillz/viskillz-blender`. The Blender project and the source code is published under *Apache-2.0 license*.

2.4.3. Relevant Theses

The results of this section were published in two papers: the original script-aided method was published in journal *Annales Mathematicae et Informaticae*. The journal was ranked as Q3 in 2021. The extended method was published in journal *SoftwareX* of publisher Springer. The journal had an impact factor of 3.4 and was ranked as Q2 in 2022.

Thesis 1

We designed 205 shapes and 31 cutting planes manually in Blender. These shapes are derived from the classic Mental Cutting Test exercises and can be used to analyze the shapes and develop scripts to yield assets.

Journal papers:

- R. Tóth, 2021
- R. Tóth, B. Tóth, Miklós Hoffmann, and Zichar, 2023

Thesis 2

We designed a script-aided method that applies predefined permutation factors on the manually designed meshes and cutting planes to yield a great number of Mental Cutting Test scenarios.

Journal papers:

- R. Tóth, 2021
- R. Tóth, B. Tóth, Miklós Hoffmann, and Zichar, 2023

3. Post-processing MCT assets

3.1. Introduction

After creating the method for permuting scenarios, our research focused on developing an open-access database to provide automatically permuted resources for users who want to deal with MCT exercises, such as students, instructors, and researchers. Solutions for multiple questions should be found and implemented to create the database and compose exercises from individual scenarios.

- *How can multiple scenarios be compared to each other?*
  First, the factors of the permutation can describe the relationship between a given pair of scenarios. Second, the shapes of the intersections can also be compared to each other.

- *How can we enhance the script-aided method?*
  Now, the instructors mostly deal with reviewing the automatically generated scenarios. The filtering of the scenarios depends on two parameters: the 2D projection of the scenario and the shape of the answer. If both of them fulfill a set of criteria, a scenario can be accepted; otherwise, it should be dropped. The method can be enhanced with automatized decision-making; an image processing algorithm can be designed, which task is to analyze the intersections and determine whether they contain anomalies or not. In that case, instructors have to deal only with the 2D projections, and abnormal intersections can be automatically detected.

- *How can the difficulty of an exercise be determined*?
  Based on previous experience (Németh, Sörös, and Miklós Hoffmann, 2007), one factor is the answers' shape and morphologic features. Thus, the correct representation of the answers not only can tell the similarity between multiple answers, but it is also possible to retrieve information on the difficulty level from them.

- *How can similar answers be detected?*
  The most trivial answer to this question is comparing their morphological representation. Multiple answers can be considered identical regardless of their orientation and minor differences; humans cannot realize that.

- *How can alternative answers be offered automatically?*
  Our vision is that different acceptable methods can be found to construct the set of answers derived from image processing, decision-making, and data mining. Multiple algorithms (such as clustering and different decision-making systems) can be executed on the morphological features of the intersections, yielding acceptable output.

- *How can we reduce the size of assets?*
  The size of a large data set containing redundant assets can be significantly decreased based on the output of the matching algorithms, encoding each unique asset only once.

Most questions require the processing and comparison of the automatically generated intersections. Thus, we checked thousands of intersections manually and found that the output of Blender contains anomalies in many cases, which change the basic morphological features of the corresponding intersections. Based on our manual validation of these intersections, we designed an automatized approach that is capable of detecting errors in our intersections, then dropping or correcting the corresponding scenarios – based on the question of whether the error only changes the representation of the intersection or makes the user unable to give the correct answer to the offered exercise. Finally, we implemented a vector-based matching for intersections to yield a unique set of shapes for constructing exercises, including their possible answers. The previously performed steps of modeling and permutation and the further post-processing and matching of assets can be seen in Figure 3.1.

3.2. Resolving errors in intersections

As the script-aided method deals with a great number of automatically generated images, it is necessary to validate the correctness of the output – in other

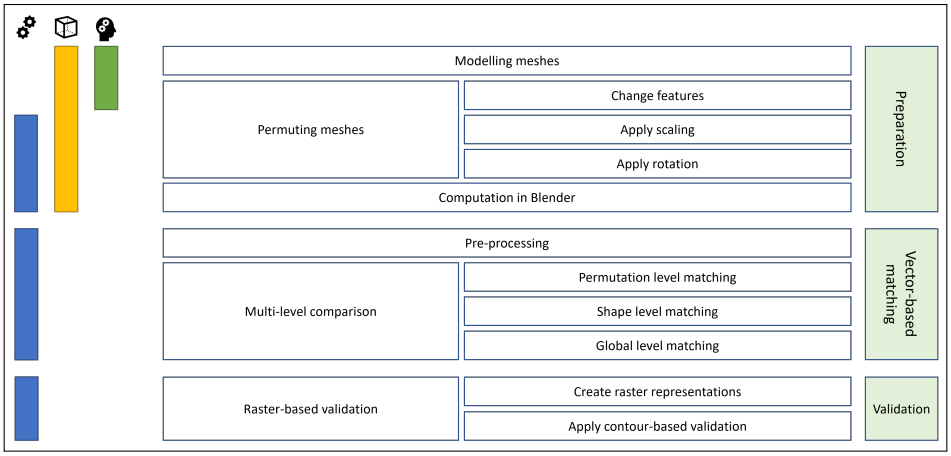


Figure 3.1: The post-processing of MCT assets has three stages: its pre-processing step eliminating the too-short edges from intersections, a divide-and-conquer vector-based matching, and its raster-based validation. These stages do not need human contribution.

words, we should check each automatically generated intersection and determine whether its scenario can be offered as an assignment or not. Thus, a post-processing algorithm should be designed and implemented, using which common features of intersections can be extracted, and their scenarios can be dropped or kept as a verdict of a manual or an automatized decision progress. The post-processing method can be done using two different approaches:

1. As Blender's FreeStyle SVG Exporter is capable of rendering SVG documents, the paths of assets generated by Blender can be parsed, processed, and manipulated. In Python, it can be easily done with the `minidom` (Python Software Foundation, n.d.) library.

2. The coordinates of vertices (and their corresponding edges) can be retrieved using Blender's Python API. This method gives an opportunity to avoid the time-consuming rendering process using Blender but makes it possible to use the raw coordinates during the computations. We choose this method because this method guarantees a faster way to perform coordinate-based computations. On the other hand, the JSON specification does not limit the number of precision digits in float values so that the values can be exported with more digits than the number Blender uses in its default configuration.

3.2.1. Common errors

The concept of the validation process is designed on the experience of randomly, manually performed validation of previously rendered scenarios. During that period, we found the following abnormal features in the data set:

1. **Noise vertices and edges** – Too short edges and their corresponding vertices appear in many scenarios. We found that these edges can be derived from multiple situations.

   (a) **Real edges**. Many scenarios contain quite small edges that can be easily considered noise. It is hard to manage them with any algorithm: usually, noise-generated edges are longer than the finest features of the intersections. In the iterative algorithm, various manually designed meshes are getting intersected with multiple cutting planes, and their combinations sometimes result in really small regions or lakes with their corresponding edges. Moreover, the contour of a larger region can also contain small edges due to the same purpose. As we mentioned, these edges are correct; on the other hand, it is hard to differentiate them from noise in algorithms, as humans cannot detect and manage them due to their size. Thus, these intersections can be dropped or can be kept; the decision depends on various factors, such as the level of difficulty that the instructors want to achieve with the selected scenarios.

   (b) **Noise edges**. Another pattern can be detected in the set of manually checked shapes: edges having really small lengths appear near the junctions of other edges, adding extra vertices and edges to the representations. The real purpose of their appearance is unclear in most cases, but they are derived from some kind of miscalculations; thus, they should be eliminated from the representations. We guess that the main purpose of their appearance is the precision error of the IEEE 754 float type, with the use of which each coordinate is represented. Parallel, we found that many of the manually entered literals on Blender's UI immediately change, and an error appears near the 5th or 6th floating digit. In addition, instructors can only make minor precision errors while designing the

shapes. Finally, calculations of intersections – that are performed by Blender's built-in operators and several transforms should be applied to them during the process – deal with non-precise coordinates, which also results in precision errors in the shapes of the intersections. Finally, the built-in operators also work with float types, producing precision errors. Thus, it is obvious that precision errors appear in the output, and it is easier to detect and resolve them than to manually design the corresponding assets without the help of a script-aided method.

2. **Shapes that are sliver polygons** – Many combinations of cutting planes and 3D meshes result in intersections that contain very short edges; the previous point dealt with that case. However, in rare situations, the intersection can also contain a sliver region or sub-face, in which the ratio of its area and perimeter is small. In such cases, the cutting plane just touches the mesh's edge, resulting in a sliver polygon as their intersection. Consequently, multiple edges overlap, resulting in an abnormal representation. Therefore, these intersections should be omitted from the data set.

3. **Subdivided edges** – Due to the triangulated faces and the behavior of the intersecting method, most of the edges are subdivided into multiple smaller edges, also referred to as segments. While it is a natural feature of the intersection, it is required to reconstruct the original representation of each intersection, merging the corresponding segments into one edge. However, it is not as easy as it seems to be; we should find a method that handles the noise edges correctly in these scenarios too. Merging the segments is required to reconstruct the original morphological properties of the intersection.

Based on the features of the common errors, it is possible to assign each error to one of the following groups:

1. Errors that can be detected and corrected; after that, the corresponding scenario can be served as an MCT exercise or included as a possible answer.

2. Errors that are derived from the incorrect choice of cutting plane and mesh; thus, the combination cannot be used in an MCT exercise or included as a possible answer.

We must design a deterministic algorithm to make decisions over the affected scenarios and eliminate or correct them before creating a data set of MCT exercises. In this section, we introduce the stages of the proposed method, which is capable of automatically detecting and correcting the errors of the computed intersections.

3.2.2. Removing short edges

As most of the problems are derived from too short edges – never mind whether the error can be assigned to the first or second group of errors –, we should design a method to detect and eliminate them from an intersection represented by a set of edges. We found empirically that a cleaning step should be performed on the data set, in which all the edges are being removed whose length is less than $\varepsilon_l = 0.22$ using the REMOVE-NOISE-VERTICES() function of Algorithm 3.1. Note that coordinates of shapes in our representation are taken from the interval $[-10.0, 10.0]$ on both of the axles; thus, an edge having a length of 0.22 is 100 times smaller than the edges of a square having parallel edges to the coordinate axles. This means that only 1% of the possible lengths is classified as noise.

3.2.3. Visualizing edges

In this step, we generate multiple SVG representations for each intersection, which makes us able to check whether the cleaned list of edges can be accepted or not. Additionally, different types of errors can be detected using the representations; thus, we can classify the errors before the next correction

Algorithm 3.1 Removing too short edges

```
1: function REMOVE-NOISE-VERTICES(E, ε_l)
2:     for each e ∈ E do
3:         if SAME-VERTICES(e[0], e[1], ε_l) then
4:             remove e from E
5:         end if
6:     end for
7: end function
```

step. First, we used these resources in a manual validation process, in which we classified the common errors and designed the decision process. Additionally, they can be used as an input of an auto-correction algorithm, which reconstructs the correct representation of the shape, fixing the appeared errors. These representations are the following:

1. *The black shape of the answer.* The contour of the shape is represented by black lines.

2. *Thick, colored shape of the answer.* The contour of the shape is represented by thick, colored lines. A unique color belongs to each edge.

3. *Thin, colored shape of the answer.* The contour of the shape is represented by thin colored lines. A unique color belongs to each edge.

These representations are generated from the list of coordinates, totally independently from the use of Blender. The corresponding Python script uses package `minidom`, and the structure of the encoding strictly applies the same transforms on the coordinates that would be performed in the Blender-side rendering process. In the next step, we create the raster representation of each SVG document with package `cairosvg` (Ayoub, n.d.) to support the pixel-wise processing of the output. The representations are created with the following features:

1. In the case of the `black` representation, we use the original, raw coordinates of shapes; there was no additional transform performed on the coordinates. Thus, each line segment is rendered in its initial position.

2. In the case of the `thin` and `thick` representations, we use the cleaned set of edges to form each intersection. Additionally, the common transforms are applied on the edges – the shape is scaled to an identical size, centered both horizontally and vertically in the viewport. On the other hand, due to the elimination of short edges, their segments may not form a continuous series of line segments, making us able to compare them to the black skeletons.

First, a human validation was performed on these data sets, using special PDF sheets generated with the use of Pandoc and its Pandoc-flavoured Markdown syntax. Each page contained exactly one scenario, with its unique ID, the expected number of edges (the length of the edges describing the intersection after the cleaning step), and all three representations. In the case of `thin` and `thick`, the black contour was also generated into the same asset under the colorful line segments. It resulted in a skeleton, which should be overlapped by the colorful segments.

3.2.4. Detecting errors

These resources can be used to detect possible errors both manually and in an automatized way. Additionally, it is also possible to determine the type of the error and decide whether it should be corrected or the scenario should be dropped. Based on the features of the representations, a shape that does not contain any error should fulfill the following conditions:

1. Each colored edge should overlap an original, black edge (both in `thin` and `thick` versions).

2. Each black edge should be overlapped by one colored edge (both in `thin` and `thick` versions).

3. The number of edges and the number of different colors used in `thick` representations should be equal.

4. No edges can be subdivided into multiple line segments, which is a consequence of the first two points.

The parallel use of `thin` and `thick` versions are needed because – due to the pre-processing step – small gaps can be found between different edges. In `thick` representations, the weight of each line fills the possible gaps around junctions. However, it is not possible to detect quite small edges if the thickness of their neighbors overlaps their appearance. Thus, `thin` representations are used to provide an opportunity to detect short edges and make us able to count the edges of all shapes.

Table 3.1 contains two scenarios, both of them contain noise edges initially. In the case of intersection `4B`, we can see a lake on its surface, formed by a triangle, represented by its three really short edges. Both of them are beyond the limit of ε, thus removed from the representation. Intersection `4A` is more interesting because we can detect only one region in it; however, the original black segments are not centered vertically. After removing all noise edges, the colored segments are moved to the center of the viewport. The purpose of that feature is that the original representation contained an extremely small region at the bottom of the viewport. By involving these coordinates in the computations, the intersection was centered both vertically and horizontally. These edges were removed before the generation of the colored segments. As a result, the colored and the black representations do not overlap each other, and the error of the scenario can be determined.

Table 3.2 contains scenarios in which all the black edges are overlapped by colored ones; however, the number of different colors does not meet the number of edges. Intersection `5A` was generated from a scenario in which the cutting plane just touches one edge of the shape, resulting in overlapping edges. Thus, the bright green line segment overlaps another edge. In the case of intersection `5B`, there is a small region formed by three line segments. However, one of them did not exceed the length of ε, so it was eliminated from the `thin` representation. Thus, the number of different colors is smaller than the number of expected edges. Finally, scenario `5C` is similar to `5A`; the dark green line segment overlaps another segment.

The differences between the representations and the features of the mesh can be detected manually. On the other hand, it is easy to automatize this process. We must generate raster images from the coordinates and perform a classic, pixel-wise processing of the intersections. The color of each pixel can be determined in both the `thin` and `thick` representations and can be com-

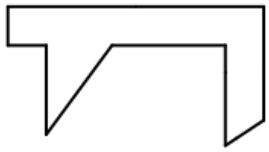
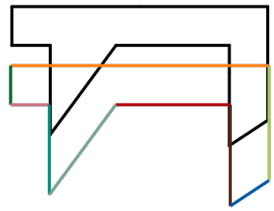
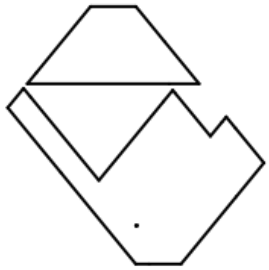
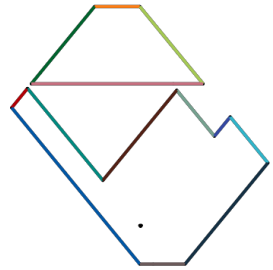
ID	black	thin
4A		
4B		

Table 3.1: Examples of scenarios that cannot be offered due to small regions or lakes. It can be seen that the `black` representation on the left side and the `thin` representation on the right side. The errors can be detected with the black pixels of representation `thin`.

pared to the expected number of edges. This eliminates the need for human operation.

3.2.5. Resolving errors

Once we are able to detect scenarios that contain errors, we must find a deterministic, coordinate-based algorithm that adjusts their edges, resulting in a correct shape:

1. We must find all the edges that are not connected to at least two other edges.

2. With an iterative algorithm, we must match these edges to form pairs: it is required to find the closest pair for each edge. The matching is performed with an upper threshold of Euclidean distance, which increases in each iteration.

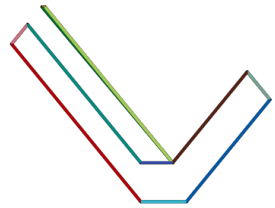
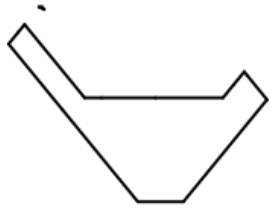
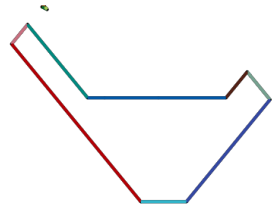
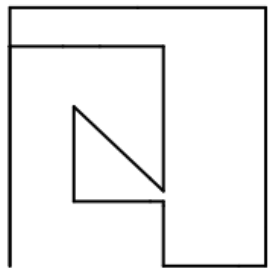
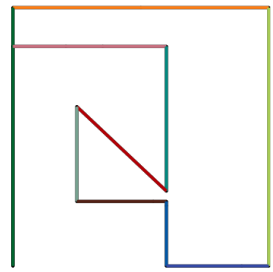
ID	black	thin
5A		
5B		
5C		

Table 3.2: Examples of scenarios that contain anomalies in their representation. These errors can be identified by counting the edges in the representation and the number of different colors that appear in representation `thin`.

3. For each pair of edges, we must calculate their intersection. Based on the intersection, two edges can be merged:

   (a) If they are parallel and the distance of their vertices is under the ε_d threshold.

   (b) If the intersection point is an endpoint of a segment, an additional third edge should be added to connect them.

   (c) If the intersection point is out of the viewport, a third edge should be added to connect them again.

3.3. Vector-based matching of intersections

As a result of the script-aided permutation method, shapes are stored with their coordinates in JSON documents. Moreover, we have already post-processed the edges and resolved the issues caused by too-short edges. In the next step, we developed a divide-and-conquer algorithm to apply a sequence of comparisons on the shapes, yielding unique sets of intersections.

3.3.1. Basic functions

The computations require elementary functions of coordinate geometry. However, those should be implemented carefully due to the floating-point errors, features, and requirements of the output's planned use cases. In this section, we introduce the most significant functions that serve as the base of comparisons during the further sections. Our implementation uses *Euclidean distance* as a metric of the distance between points. Thus, SAME-VERTICES() of Algorithm 3.2 let us able to determine whether two vertices can be considered the same or not using an ε precision error.

Algorithm 3.2 Checking whether two vertices are the same

```
1: function SAME-VERTICES(v_a, v_b, ε)
2:     return sqrt((v_a[0] − v_b[0])^2 + (v_a[1] − v_b[1])^2) < ε
3: end function
```

Comparison of edges will be used in various functions, thus we must introduce function SLOPE() defined in Algorithm 3.3, which returns the slope of a given edge using an elementary ARCTAN() function.

Algorithm 3.3 Calculating the slope of an edge

```
1: function SLOPE(e)
2:     return ARCTAN(e[1].y − e[0].y, e[1].x − e[0].x)
3: end function
```

The use of function SLOPE() makes it possible to compare two edges (e_a and e_b) and determine whether they are parallel or not using a given ε using the PARALLEL-EDGES() function defined in Algorithm 3.4. If the directions of the edges are non-obvious, we must check whether the difference between their slopes approximates 0 or π using the given ε error.

Algorithm 3.4 Checking whether two edges are parallel

```
1: function PARALLEL-EDGES(e_a, e_b, ε)
2:     Δ = ABS(ABS(SLOPE(e_a) − SLOPE(e_b)) mod π)
3:     return Δ < ε or ABS(π − Δ) < ε
4: end function
```

Using the previous functions, we can define the SAME-EDGES() function of Algorithm 3.5, which consumes two edges (ε_a and ε_b) and two error limits (ε_α and ε_d), which represent the error limit of slopes and coordinates (in that order). The function checks whether the difference between the slopes can be accepted, then cross-checks the first and second vertices of the edges using function SAME-VERTICES().

Algorithm 3.5 Checking whether two edges are the same

```
1: function SAME-EDGES(e_a, e_b, ε_α, ε_d)
2:     if not PARALLEL-EDGES(e_a, e_b, ε_α) then
3:         return false
4:     end if
5:     if SAME-VERTICES(e_a[0], e_b[0], ε_d) and SAME-VERTICES(e_a[1], e_b[1], ε_d) then
6:         return true
7:     end if
8:     if SAME-VERTICES(e_a[1], e_b[0], ε_d) and SAME-VERTICES(e_a[0], e_b[1], ε_d) then
9:         return true
10:    end if
11:    return false
12: end function
```

Our matching algorithms strongly depend on basic transformations of shapes, edges, and their vertices. Thus, we introduce the most important geometric transformations in Algorithm 3.6. Functions ROTATE-VERTEX() and

ROTATE-SHAPE() yield the rotated coordinates of a vertex and a shape (interpreted as a list of its edges). Functions FLIP-VERTEX() and FLIP-SHAPE() yield the horizontally flipped (mirrored) coordinates of a vertex and a shape, while functions SCALE-VERTEX() and SCALE-SHAPE() apply scaling on the given parameters. Each function deals with immutable data structures, thus not changing the state of the original vertices and edges. This is an important behavior from the view of future algorithms.

Algorithm 3.6 Elementary transformation operators of vertices and shapes

```
 1: function ROTATE-VERTEX(v, φ)                                  ▷ rotates a vertex
 2:     return {v.x * COS(φ) − v.y * SIN(φ), v.y * COS(φ) + v.x * SIN(φ)}
 3: end function
 4:
 5: function ROTATE-SHAPE(S, φ)                                   ▷ rotates a shape
 6:     return {ROTATE-VERTEX(e[0], ε), ROTATE-VERTEX(e[1], ε)} for e ∈ S
 7: end function
 8:
 9: function FLIP-VERTEX(v)                                       ▷ flips a vertex
10:     return {−v.x, v.y}
11: end function
12:
13: function FLIP-SHAPE(S)                                        ▷ rotates a shape
14:     return {MIRROR-VERTEX(e[0]), MIRROR-VERTEX(e[1]} for e ∈ S
15: end function
16:
17: function SCALE-VERTEX(v, ψ)                                   ▷ scales a vertex
18:     return {v.x * ψ, v.y * ψ}
19: end function
20:
21: function SCALE-SHAPE(S)                                       ▷ scales a shape
22:     return {SCALE-VERTEX(e[0]), SCALE-VERTEX(e[1]} for e ∈ S
23: end function
```

Finally, we define function SAME-SHAPES() in Algorithm 3.7, which consumes two shapes (s_a and s_b), and the common epsilon values (ε_d and ε_α). Two shapes can be classified as the same if and only if they contain the same number of edges. Thus, a pairwise comparison should be performed between their edges. In our implementation, a duplicate called s_c is created from s_b, then we iterate through the elements of s_a. For edge e_a of s_a, we must find a corresponding e_c in s_c using the given ε values. If an acceptable edge is found, we remove it immediately from s_c. If no corresponding edge was found for e_a

in s_c, the shapes cannot be the same since we cannot finish matching their edges.

Algorithm 3.7 Determining whether two shapes are the same

```
1: function SAME-SHAPES(s_a, s_b, ε_d, ε_α)
2:     s_c ← copy of s_b
3:     for each e_i ∈ s_a do
4:         deleted ← false
5:         for each e_b ∈ s_c do
6:             if SAME-EDGES(e_a, e_c, ε_d, ε_α) then
7:                 remove e_c from s_c
8:                 deleted ← true
9:             end if
10:        end for
11:        if not deleted then
12:            return false
13:        end if
14:    end for
15:    return true
16: end function
```

3.3.2. The algorithm

Shapes are processed in their initial order, and our goal is to construct a set of unique shapes (S) by adding shapes to the initially empty set. For each shape, we must check whether S already contains the shape or not. This requires pairwise comparisons performed between the current and all previously added unique shapes. Additionally, there is no guarantee that the shapes of the current comparison have the same orientation; thus, the continuous use of rotation and scaling operators is required. As Table 3.3 shows, the permutation yields more than 150,000 shapes. To perform the comparison between these shapes in an acceptable run-time limit, we introduce a divide-and-conquer approach and define functions that can be used at multiple levels using different parameters.

The base of the comparisons is function FIND-SHAPE of Algorithm 3.8 that gets six parameters: the initial orientation of the current shape (s_o), the set of previously selected unique shapes (S), the set of rotation offsets (Φ), the scaling ratio (ψ), and the epsilon values with their common notation (ε_d and ε_α). The current shape is rotated with each $\phi \in \Phi$ offset, then scaled to the uniform size using ratio ψ. The edges of the rotated and scaled shape are ordered again

Group	Models	Correct	Empty	Group	Models	Correct	Empty
#01	10	7,680	720	#15	10	7,176	1,224
#03	10	7,398	1,002	#16	10	7,666	734
#05	8	6,144	576	#17	10	7,692	708
#06	8	6,004	716	#18	10	7,578	822
#07	10	7,680	720	#19	10	7,570	830
#09	10	7,230	1,170	#20	10	7,636	728
#10	10	7,679	720	#21	10	7,644	756
#11	11	8,140	1,100	#22	10	7,536	864
#12	10	7,462	936	#23	10	7,680	720
#13	8	5,920	800	#24	10	7,680	720
#14	10	7,680	720	Total	205	154,875	17,286

Table 3.3: Groups with the number of manually permuted models, correct (non-empty, non-asserted) and empty intersections.

using the previously introduced method; then, the matching is performed on set S. If we find a similar shape to s_a in S, then s_a is not unique; otherwise, s_a is classified as a unique intersection and should be added to S. Note that the horizontally flipped (mirrored) transformation of s_a is also used in the comparisons. Figure 3.2 shows how different orientations of the current shape and its flipped version can be compared to the previous unique shapes.

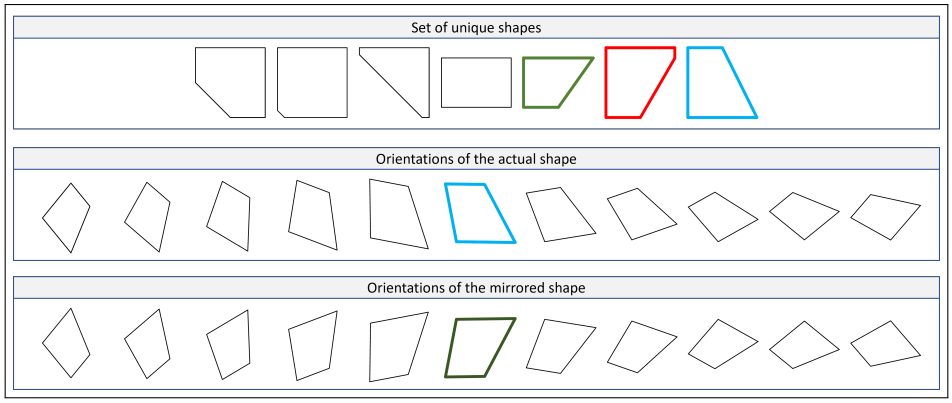


Figure 3.2: Elements of the top group are previously recognized unique shapes, while the elements of the middle and bottom groups are different orientations of the current shape and its flipped version. Blue and green shapes are the most similar to each other, as they have two parallel edges, and their total number of edges is the same. The red shape cannot be compared to the current one since their number of edges differ.

Algorithm 3.8 Matching shapes using their coordinates

```
1: function FIND-SHAPE(s_o, S, Φ, ψ, ε_d, ε_α)
2:     for each φ ∈ Φ do
3:         s_a ← SCALE-SHAPE(ROTATE-SHAPE(s_o, φ), ψ)
4:         ORDER-SHAPE(s_a)
5:         i ← CONTAINS-SHAPE(S, s_a, ε_d, ε_α)
6:         if i ≠ null then
7:             return i
8:         end if
9:     end for
10:    return null
11: end function
12:
13: function CONTAINS-SHAPE(S, s_a, ε_d, ε_α)
14:    s_m ← FLIP-SHAPE(s_a)
15:    for each s_i ∈ S do
16:        if SAME-SHAPES(s_a, s_i, ε_d, ε_α) or SAME-SHAPES(s_m, s_i, ε_d, ε_α) then
17:            return i
18:        end if
19:    end for
20:    return null
21: end function
```

We execute the previously introduced matching algorithm in three stages to form a divide-and-conquer algorithm. First, we compare only shapes that are retrieved from the same mesh. That results in a local comparison, with at most 120 intersections that can be redundant if the mesh has symmetric features. Next, the unique intersections of the first stage are merged into one set of unique shapes per mesh group. If different manually designed permutations of an original mesh are similar, this step also removes a huge amount of redundancy. The final stage merges the set of each mesh group into one global set of unique shapes.

3.3.3. Permutation level matching

Table 3.4 contains an overview of the first stage of the matching algorithm. 172,200 shapes could be generated using 21 meshes, five cutting planes, 24 rotations, and seven scaling vectors. However, 10% of the combinations result in empty intersections. On the other hand, 39 intersections were dropped because we could not reconstruct their edges from the line segments. Finally, 399,988 intersections are classified as unique in their local environment, which

is 23.22% of the total number of permutations and 25.82% of the non-empty and valid shapes. Figure 3.3 shows that most of the matches were found using $\varphi = 0°$ and $\varphi = 180°$ rotation offsets. Note, that comparisons were performed in the order of $\varphi = 0°$, $\varphi = 90°$, $\varphi = 180°$ and $\varphi = 270°$. Thus, the performance of the comparisons could be improved by changing the order of the Φ vector's members. Furthermore, it is important to mention that the matching was also performed using 360 different φ offsets, and the output was the same as the output of the previous set.

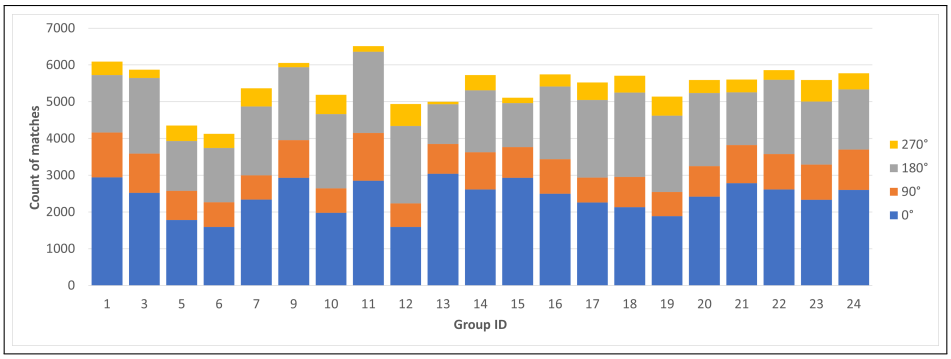


Figure 3.3: The number of matches using different φ offsets in each group.

3.3.4. Shape level matching

The unique shapes of the previous stage – a total of 399,988 shapes – were processed in this stage, resulting in 21 sets of unique shapes. In this stage, the matching function was called with the same parameters as in the previous stage, except for the Φ set of offset vectors. Now Φ contains each integral offset measured in degrees from the interval $[0,359]$, making it possible to find all possible matching shapes. Even though this step requires performing more comparisons, we can expect that most of the matching φ offsets are elements of set $\{0°,90°,180°,270°\}$ again. Thus, moving these φ offsets to the first positions of vector Φ results in fewer comparisons using a simple, greedy approach. Note that we used the previously pre-processed data set without the need for further pre-processing. Table 3.4 shows that almost half of the locally unique shapes are classified again as unique shapes at the stage of groups, resulting in a total number of 18,888 unique shapes from the set of 399,988 shapes. Figure 3.4 compares the number of matches found (in a logarithmic

Pre-processing			1st stage			2nd stage		3rd stage	
Group	Total	Correct	Unique	Ratio (T)	Ratio (C)	Unique	Ratio	Unique	Ratio
#01	8,400	7,680	1,588	18.90%	20.68%	348	21.91%	348	100.00%
#03	8,400	7,398	1,524	18.14%	20.60%	533	34.97%	458	85.93%
#05	6,720	6,144	1,791	26.65%	29.15%	935	52.21%	875	93.58%
#06	6,720	6,004	1,877	27.93%	31.26%	1,001	53.33%	897	89.61%
#07	8,400	7,680	2,312	27.52%	30.10%	1,427	61.72%	1,415	99.16%
#09	8,400	7,230	1,176	14.00%	16.27%	445	37.84%	245	55.06%
#10	8,400	7,679	2,491	29.65%	32.44%	1,682	67.52%	1,589	94.47%
#11	9,240	8,140	1,628	17.62%	20.00%	369	22.67%	336	91.06%
#12	8,400	7,462	2,525	30.06%	33.84%	1,048	41.50%	1,006	95.99%
#13	6,720	5,920	920	13.69%	15.54%	363	39.46%	287	79.06%
#14	8,400	7,680	1,955	23.27%	25.46%	795	40.66%	577	72.58%
#15	8,400	7,176	2,068	24.62%	28.82%	787	38.06%	661	83.99%
#16	8,400	7,666	1,921	22.87%	25.06%	1,126	58.62%	944	83.84%
#17	8,400	7,692	2,167	25.80%	28.17%	934	43.10%	856	91.65%
#18	8,400	7,578	1,873	22.30%	24.72%	1,162	62.04%	975	83.91%
#19	8,400	7,570	2,428	28.90%	32.07%	1,026	42.26%	923	89.96%
#20	8,400	7,636	2,048	24.38%	26.82%	1,155	56.40%	999	86.49%
#21	8,400	7,644	2,040	24.29%	26.69%	873	42.79%	685	78.47%
#22	8,400	7,536	1,679	19.99%	22.28%	783	46.63%	628	80.20%
#23	8,400	7,680	2,092	24.90%	27.24%	1,082	51.72%	826	76.34%
#24	8,400	7,680	1,905	22.68%	24.80%	976	51.23%	347	35.55%
Total	172,200	154,875	40,008	23.23%	25.83%	18,850	47.12%	15,877	84.23%

Table 3.4: Counts of the total, empty, dropped, and unique intersections that belong to each model group. The last column contains the ratio of unique intersections, calculated with the number of total and non-empty, and non-dropped intersections.

scale) with the corresponding φ offsets. Most of the matches found again using $\varphi \in \{0°, 90°, 180°, 270°\}$.

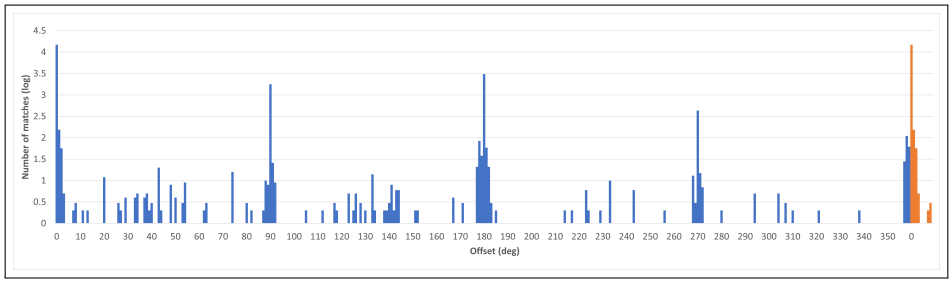


Figure 3.4: The number of matches using different φ offsets in a logarithmic scale.

3.3.5. Global level matching

In the last stage, we use 360 different φ values again, ordering $\varphi \in \{0°, 90°, 180°, 270°\}$ offsets again to the first positions of vector Φ. Table 3.4 contains the number and ratio of unique shapes, while Figure 3.5 represents data about the matches and offsets. The figure has the same features as Figure 3.4; thus, the matching algorithm has the same behavior both on the shape and global levels.

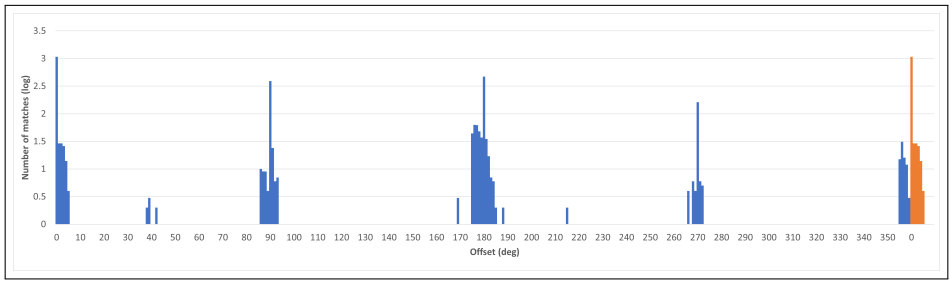


Figure 3.5: The number of matches using different φ offsets in a logarithmic scale.

3.4. Raster-based validation of the matching

Finally, we give a short description of a possible method that can be used to validate our vector-based matching algorithm, calculating the value of its *accuracy* by using a raster-based approach. Note that the comparisons used coordinates, slopes, and the number of edges throughout the vector-based implementation. In the validation, we eliminate all these parameters by representing all the shapes with one-channel 200*200 pixel-sized matrices, having normalized intensities (1.0 in foreground pixels, 0.0 in background pixels). To provide a base for the precise comparison of raster images, we used Python's `cairosvg` library, which was built over the popular *GTK Toolkit*, also used by favored image editors such as *GIMP* and *Inkscape*. We should note that SVGs generated with Blender's *FreeStyle SVG Exporter* plugin contain non-deterministic extra edges in shapes having holes. Thus, using coordinates yielded from the low-level representation of each shape should be used to yield the outlines of each shape, and the needed post-processing of the images requires only a few basic image-processing steps.

The precise one-layer contours of each shape (which is more accurate than the original one, derived from the SVG strokes) are used to support the matching between images. For inner contours, we use 8-connectivity (thus, pixels are neighbors if they have at least one common vertex), while for outer contours, we use 4-connectivity (thus, pixels are neighbors if they have at least one common edge). In that approach, we classify two shapes as identical if the single difference is in their contours. Thus, we can introduce the so-called region of interest (*ROI*), which is the union of the inner and outer contours of the two shapes being compared. The comparison of the raster representations is derived from implementing the vector-based function. Thus, the current shape is simultaneously scaled and rotated over the previously selected unique shapes. The matching function gets different binary representations and contours of the current shape in different orientations (B_a and C_a), and the stored binary representation and contour of a unique shape (b_u and c_u). The algorithm compares each orientation of the current shape to the unique intersection, then stores the matrices representing their differences in list Δ. Next, we select the difference matrix with index i_{min} that contains the least number of values 1. During the comparison, the region of interest (*ROI*) is the union of the contours of the current and the unique shapes. Finally, the function checks whether or not matrix δ contains only 1 as a value in the *ROI*. A match is found if the two shapes differ only in the *ROI*.

Although the validation of unique (positive) and non-unique (negative) shapes differ, they rely on the comparison function. When processing a new shape, we should compare it to all previously marked as *unique*. Meaning that every shape is rotated with each offset from the interval $[0°, 90°]$, yielding 90 different orientations of the current shape. After that, the representation of each previously recognized unique shape is compared to each orientation. The shape should be considered a non-unique, false positive case if we find a match. If no match is found, the function returns a final value of *true*. In the case of non-unique (negative) intersections, we have extra knowledge about the matching that was found by the coordinate-based algorithm (parameters i_m and φ_m). Thus, only φ_m and only one previously processed shape should be involved in the comparison, having index i_m. If we detect that the match is acceptable between the shapes, the current shape should be considered a non-unique (false negative) case. Otherwise, we can check whether a match

exists between the current shape and other previously processed shapes. This can be done by re-using the validation of unique (positive) cases, but the returned value is negated. This method lets us check the accuracy of the previously introduced vector-based matching algorithm. However, the algorithm has a large time, and memory complexity, as about 400 million comparisons are needed to validate the output of the first locally performed stage founding 399 988 unique shapes from the set of 172,200 shapes. The amount of the needed comparisons can be approximated with the following cases:

1. If the status of a shape is *TP*, we needed to compare its area with the areas of all previous, unique shapes. It is already a simpler mechanism than comparing the shape to all previous shapes. This method was performed in the case of 38,666 shapes.

2. If the status of a shape is *TN*, we needed to compare its area with the area of its alternate shape. However, matches are not always acceptable: in this case, another matching shape should be found before the shape is classified as *FN*. This method was performed for 230 shapes.

3. If the status of a shape is *FP*, all previously classified unique shapes should be compared to the current shape. This step was performed in the case of 1,361 shapes.

4. If the status of a shape is *FN*, an iteration had been started at first to index, and the matching was applied on all unique shapes before finding the first match. This step was performed in 237 cases.

Table 3.5 and Table 3.6 contain the result of the data set's validation, aggregated to the level of mesh groups. A total of 154,875 intersections were involved in the comparison (not counting the shapes dropped by the assertion and the empty cases), from which the comparison finds 38,666 *true positive* cases and 114,650 *true negative* cases. Thus, the contour-based comparison of the shapes reveals that the accuracy of the vector-based matching is 98.97%. Both algorithms also process intersections containing anomalies; zero non-empty and non-asserted shapes were excluded from the matching.

Group	Total	TP	TN	FP	FN	ACC
#01	7,680	1,479	6,092	109	0	98.58%
#03	7,398	1,479	5,874	45	0	99.39%
#05	6,144	1,731	4,353	60	0	99.02%
#06	6,004	1,860	4,127	17	0	99.72%
#07	7,680	2,280	5,368	32	0	99.58%
#09	7,230	1,125	6,054	51	0	99.29%
#10	7,679	2,468	5,188	23	0	99.70%
#11	8,140	1,543	6,512	85	0	98.96%
#12	7,462	2,464	4,937	61	0	99.18%
#13	5,920	792	5,000	128	0	97.84%
#14	7,680	1,845	5,723	110	2	98.54%
#15	7,176	1,960	5,107	108	1	98.48%
#16	7,666	1,878	5,745	43	0	99.44%
#17	7,692	2,075	5,525	92	0	98.80%
#18	7,578	1,814	5,705	59	0	99.22%
#19	7,570	2,355	5,142	73	0	99.04%
#20	7,636	1,983	5,587	65	1	99.14%
#21	7,644	1,947	5,604	93	0	98.78%
#22	7,536	1,640	5,857	39	0	99.48%
#23	7,680	2,050	5,588	42	0	99.45%
#24	7,680	1,822	5,775	83	0	98.92%
Total	154,875	38,590	114,863	1,418	4	99.08%

Table 3.5: Result of the validation of the first stage.

3.5. Remarks

The validation of the set containing locally unique shapes (the first stage output) required up to 30 hours of continuous execution on an AMD Ryzen 7 4800H processor. Four consoles were used to run the matching parallel. Meanwhile, the *Numba* library was also used to introduce more efficient, automatically paralleled C++ codes instead of the lazier Python implementations. The used notebook had a total of 8 GB memory, 70-90% of which was in use during the matching. We can assume from the previous chapters that the matching can be performed with the same parameters on the vector-based approach at multiple levels. Thus, we can assume that each level's accuracy should be around 98.9% without applying validation, which requires much more resources.

Group	Total	TP	TN	FP	FN	ACC
#01	1,588	316	1,138	32	102	91.56%
#03	1,524	523	826	10	165	88.52%
#05	1,791	890	818	45	38	95.37%
#06	1,877	986	787	15	89	94.46%
#07	2,312	1,402	855	25	30	97.62%
#09	1,176	435	665	10	66	93.54%
#10	2,491	1,671	740	11	69	96.79%
#11	1,628	336	1,239	33	20	96.74%
#12	2,525	1,010	1,282	38	195	90.77%
#13	920	343	515	20	42	93.26%
#14	1,955	763	1,089	32	71	94.73%
#15	2,068	764	1,238	23	43	96.81%
#16	1,921	1,120	709	6	86	95.21%
#17	2,167	915	1,224	19	9	98.71%
#18	1,873	1,140	664	22	47	96.32%
#19	2,428	1,003	1,289	23	113	94.40%
#20	2,048	1,102	825	53	68	94.09%
#21	2,040	858	1,119	15	48	96.91%
#22	1,679	757	838	26	58	95.00%
#23	2,092	1,008	906	74	104	91.49%
#24	1,905	933	880	43	49	95.17%
Total	40,008	18,275	19,646	575	1,512	94.78%

Table 3.6: Result of the validation of the second stage.

We must mention that the use of Windows Defender can affect the execution time of scripts that write or read a great number of assets to or from the disk. However, in the case of the matching algorithm, we used only a few JSON documents. Thus, Windows Defender did not impact the measured times as it will do in the case of the encoding scheme of the next section.

3.6. Summary

3.6.1. Results

This chapter introduced our post-processing steps that should be performed on the dataset retrieved from Blender. The proposed algorithms are related to two different directions. First, data retrieved from Blender contain various anomalies that can be derived from the scenarios and the computational errors of Blender or the insufficient combination of the cutting plane and the 3D shape. These anomalies change the description of a shape, making their processing comparison impossible during the exercise assessment. Second, the set of unique intersections must be obtained to construct exercises from scenarios in the further steps. Thus, we designed a divide-an-conquer algorithm to select the unique shapes from the set of intersections.

3.6.2. Relevant thesis

The result of this chapter was partially published in our conference paper, presented at *The 20th international Conference on Geometry and Graphics (ICGG 2022)*.

Thesis 3

We post-processed the output of the permutation script and found anomalies in the SVG assets. We designed and implemented an automated raster-based validation process based on our experience with human validation. Assets having errors due to calculation errors can be corrected, or their corresponding scenario can be dropped automatically.

Conference paper: R. Tóth, B. Tóth, Zichar, et al., 2023a

4. Encoding GLB assets of MCT scenarios

4.1. Introduction

The dataset with the recently implemented permutation steps contains more than 1 million (1,067,640) scenarios, including small redundancy due to the symmetrical features of the shapes. Focusing on the GLB models for supporting AR, VR, and 3D developments, the assets have a total size of 6,521,971,888 bytes, but the needed disk storage is more than 8 GB in a Windows 11 operating system. Therefore, filtering and processing assets or constructing exercises manually or automatically in an application requires storing the dataset in the database. Thus, a method should be found to allow developers to compress the dataset, making us able to serve content in real-time. One possible solution is to find existing compression algorithms; in the case of GLB assets, Draco compression[5] is a well-known and universal method to compress the assets.

However, different features of the permutation algorithm should be detected in the assets. With the design and development of a domain-specific algorithm, it is possible to achieve a better compression ratio. In this section, we describe the common features of the assets and introduce an encoding scheme that consists of four levels.

4.2. Creation and structure of the graphical assets

In this technical section, we introduce the original GLB documents. We focus on the JavaScript Object Notation (JSON) chunk and its properties since this part of the document contains all the properties describing its data chunk. We give a short code snippet for each property, then discuss which values are required or can be omitted in the encoding scheme since they are optional features containing metadata.

Before explaining the JSON chunk, we must mention an unexpected behavior of Blender. If a user creates a custom object or opens a new project and uses the default cube, it does not have a texture or material. However, Blender adds a UV map to each object by default, which behavior cannot be changed. We believe that this behavior affects most of the assets generated us-

[5]Available: `https://google.github.io/draco/` (accessed: 13th August 2023)

ing Blender, and it is hard to detect. In this case, the list of textures is empty, and it is not possible to remove the UV map of multiple objects on the UI; designers should adjust each object separately, which takes great effort. This is important because Blender generates a texture for each object, resulting in an empty UV map. It produces a significant overhead since the average size of the related data takes 20–25% of our models (resulting in a total size of 5,070,079,296 bytes without the UV map). Thus, the rest of this paper deals with two scenarios:

- Each model has the UV map due to the behavior of Blender;
- The UV map of each model was removed before the processing.

Our assets were generated with Blender version 3.3. Thus, the description and the calculations are based on the output of this version.

4.2.1. Properties of the JSON chunk

Property `asset`

The first property of each JSON document is called the `asset`, which contains the general attributes of the actual asset, such as its version number. Our assets follow the 2.0 version of the specification (The Khronos® 3D Formats Working Group, 2021), and Blender also encodes the name of the generator tool (see Figure 4.1). Of course, these values are globally identical; none of the assets store different values in these properties. On the other hand, the property `generator` can be omitted from the documents since this optional attribute does not provide any necessary and meaningful information for the processors.

```
"asset": {
 "generator": "Khronos glTF Blender I/O v3.2.43",
 "version": "2.0"
}
```

Figure 4.1: Property `asset` of the JSON chunk in an MCT asset.

Properties `scene` **and** `scenes`

These properties describe the scenes of the asset and their hierarchy. Figure 4.2 shows that our assets contain a single scene, whose index is stored in the `scene` property. The specification follows the most practical and classic indexing, which refers to the first element of an array with an index of 0. Thus, the index of our scene is a constant value of `0`.

The property `scenes` describes each scene; thus, this property is a singleton list in our assets. Its first and only element is a `Scene` object that describes its name and refers to its corresponding node with its ID. The `nodes` property has a constant value of `[0,1]`, since each scenario contains a 3D mesh and the representation of the 2D cutting plane. Additionally, they are also being encoded in a strict order. The `name` attribute is optional, and its value is always the *Scene* literal. However, they are optional metadata that do not help the parsing and rendering methods of an asset.

```
"scene": 0,
"scenes": [
  {
    "name": "Scene",
    "nodes": [0, 1]
  }
]
```

Figure 4.2: Properties `scene` and `scenes` of the JSON chunk in an MCT asset.

Property `nodes`

This property describes the nodes of the asset, which can be interpreted as objects in Blender's terminology.

In Figure 4.3, the first object of the array describes the cutting plane, while the second object represents the 3D mesh. The order of the elements — based on the relationship of the corresponding objects in Blender — can be different. However, the naming convention of intersection frames (having IDs starting with `R`) and 3D meshes (having IDs beginning with the `Classic` prefix) lets us easily distinguish them in any document. Furthermore, there must be an identical prefix assigned to the cutting planes. Then, the meshes can be named casually. In the case of a document in which the 3D mesh is the first element of

the array, we can apply a swap operation on the array, resulting in deterministic order in all the documents. The constant value of the previous `scenes` property occurred for the same reason.

```
"nodes": [
  {
    "mesh": 0,
    "name": "R31",
    "rotation": [-0.65, 0.65, -0.27, 0.27],
    "translation": [0.5, 0, 0.5]
  },
  {
    "mesh": 1,
    "name": "Classic.0109.011",
    "rotation": [0.70, 0, -0.70, 0]
  }
]
```

Figure 4.3: Property `nodes` of the JSON chunk in an MCT asset.

Each object starts with the property `mesh`, which refers to the corresponding mesh object with its ID. The order of the `Node` and `Mesh` objects strictly follow the same order in their arrays. Thus, each of these properties contains the index of the actual `Node` object (this feature will also be considered in the description of properties having similar sequences in the rest of the document). It is followed by the `name` property, which contains the name of the corresponding Blender object. They are optional metadata again, or which they do not need to be encoded.

The rest of the properties contain more useful information since the local transforms of each node are listed in these properties. If a transform was not applied in Blender, the `matrix_world` property of the object differs from the identity matrix. Thus, a GLB file contains the sequence of non-applied transformations in the order of `rotation`, `scale`, and `translation`. They are optional, but each appears if a corresponding transform should be applied to the encoded mesh. The `rotation` property contains the unit quaternions in order `(x,y,z,w)`, and the `scale` property contains the scaling factors in order `(x,y,z)`. In contrast, property `translations` contain the node's translation in order `(x,y,z)`.

Property `meshes`

This property is an array of `Mesh` objects and connects the nodes to their `Accessor` objects.

Figure 4.4 shows that each `Mesh` object starts with its `name` property, which are optional metadata again. In our Blender scene, each mesh contains exactly one primitive. Thus, each `primitives` array is a singleton in the assets. The only `Primitive` object describes the mapping between the features of the mesh and the `Accessor` objects, which will tell how the corresponding binary data can be retrieved for the given features. All the values are constant for each scenario because the first four accessors describe the cutting planes, and the last four accessors describe the 3D meshes. Values of the `attributes` object and the `indices` property are derived from this feature. Identical materials are applied to our assets, encoded in values of `material` properties using their indices. These values are also deterministic. The first material belongs to the intersection frame, and the second material belongs to the 3D mesh.

Property `meshes` is the first in which we can realize the encoded empty textures since property `TEXCOORD_0` denotes the `Accessor` object, which describes the texture of a mesh. As Figure 4.4 shows, accessors #2 and #6 refer to the textures if they are set; otherwise, each mesh has only two attributes and the `indices` property (and thus only 3 `Accessor` objects).

Property `materials`

This property contains the materials of each asset. Figure 4.5 shows that we use two materials on the meshes of our assets. Various properties can be used in each material to obtain the required appearance of the corresponding meshes. We are serving our cutting planes with a simple, black material. On the other hand, we apply a light gray, metallic material to the 3D meshes; this makes the users able to detect the edges more easily, thanks to the reflections. Thus, the property `materials` has a globally identical value.

```
"meshes": [
  {
    "name": "Cube.318",
    "primitives": [{
      "attributes": {"POSITION": 0, "NORMAL": 1, "TEXCOORD_0": 2},
      "indices": 3,
      "material": 0
    }]
  },
  {
    "name": "Cube.360",
    "primitives": [{
      "attributes": {"POSITION": 4, "NORMAL": 5, "TEXCOORD_0": 6},
      "indices": 7,
      "material": 1
    }]
  }
]
```

Figure 4.4: Property `meshes` of the JSON chunk in an MCT asset.

```
"materials": [
  {
    "pbrMetallicRoughness": {
      "baseColorFactor": [0, 0, 0, 1],
      "roughnessFactor": 0.5
    }
  },
  {
    "pbrMetallicRoughness": {
      "baseColorFactor": [0.8, 0.8, 0.8, 1],
      "metallicFactor": 0.5,
      "roughnessFactor": 0.3
    }
  }
]
```

Figure 4.5: Property `materials` of the JSON chunk in an MCT asset.

Property `accessors`

This property contains the essential high-level properties of the byte buffers, such as their types, sizes (number of elements), and domain range, making processors able to read the content of the binary buffer.

Figure 4.6 shows that each `Accessor` object refers to a `BufferView` object with its ID. This value is identical since each buffer is encoded in the built-in binary chunk. The value of property `componentType` indicates the datatype of a buffer: Code `5126` denotes an IEEE 754 float type (4 bytes), while code `5123` refers to an unsigned, short integer (2 bytes). Property `type` tells the structure of the corresponding buffer: Code `VEC3` tells that the buffer contains 3-length vectors of the specified datatype, `VEC2` means 2-length vectors, while `SCALAR` means that single values are encoded.

```
"accessors": [
  {
    "bufferView": 0,
    "componentType": 5126,
    "count": 56,
    "max": [ 1.0802764892578125,
             1.0802803039550781,
             1.0802764892578125 ],
    "min": [ -1.0802764892578125,
             -1.0802764892578125,
             -1.0802764892578125 ],
    "type": "VEC3"
  },
  ...
  {
    "bufferView": 7,
    "componentType": 5123,
    "count": 30,
    "type": "SCALAR"
  }
]
```

Figure 4.6: Properties `accessors`, `bufferViews` and `buffers` of the JSON chunk in an MCT asset.

The properties `type`, `componentType`, and `bufferView` have constant values since textured assets contain eight buffers, while non-textured assets contain six buffers having the same data types in a strict order. The properties `min`

and `max` only appear in the first buffer of each mesh because it is required for a `POSITION` accessor and optional in other cases. `Accessor` objects with indices 2 and 6 refer to textures, encoding pairs of float values.

Properties `bufferViews` **and** `buffers`

Finally, these properties give a low-level description of each binary buffer. Elements of property `bufferViews` describe a sequence of bytes behind an `Accessor`, and the property `buffers` provides the size of each view.

Figure 4.7 shows properties `bufferViews` and `buffers` of our assets. Textured assets have eight `BufferView` objects, while non-textured assets have six entries in their `bufferViews` property. Our assets are specific because we do not use external buffers; thus, each feature is encoded in a single, carried, binary chunk of the assets. Thus, the properties of these objects are specific: only the value of the `byteLength` property needs to be stored since views are sequentially encoded in the binary chunk, and their offsets are deterministic. On the other hand, the `buffers` property has exactly one entry containing the total length of views in its `byteLength` property.

```
"bufferViews": [
  {
     "buffer": 0,
     "byteLength": 672,
     "byteOffset": 0
  },
  ...
  {
     "buffer": 0,
     "byteLength": 60,
     "byteOffset": 2752
  }
],
"buffers":[{ "byteLength":2812 }]
```

Figure 4.7: Properties `accessors`, `bufferViews` and `buffers` of the JSON chunk in an MCT asset.

4.2.2. Permutation-based features

We have shown that several properties can be eliminated from the encoded files due to the specific features of our assets. Thus, some properties may be omitted from encoded assets, and others can be easily reconstructed from less information. In this section, we mention some important features of our assets.

Property `nodes`

Elements of the `nodes` array describe the frame that denotes the cutting plane, then the shape, which is being intersected by the plane. The permutation step changes their features to yield all the possible combinations in the dataset:

1. Each 3D mesh is rotated using 24 different rotation vectors.
2. Each 3D mesh is scaled using seven different scaling vectors.
3. A total number of 31 cutting planes are combined with each 3D mesh. On the other hand, four different meshes represent a cutting plane. The rest of them can be yielded by applying transformation operators on the set of selected frames.

Thus, we detach the transformations of meshes from individual documents and store them in a global configuration. For example, if a cutting plane appears in multiple assets, the same features should be coded each time in the documents. Moreover, if a shape appears multiple times, its features can also be omitted from the redundant encoding. However, scaling operations were applied in Blender since the `Bisect` operator deals with local coordinates, and this step serves as a more accessible base for computations. Thus, considering the permutation algorithm, the following properties need to be extracted from documents and should be stored in a global configuration:

1. A cleaned `Node` object for each cutting plane (a total number of 31 objects);
2. A cleaned `Node` object for each rotation vector (a total number of 24 objects).

Property `accessors`

The assets contain a deterministic number of `Accessor` objects: the length of property `accessors` is 8 in the case of textured assets and 6 in the case of non-textured assets. Their order, property `type`, and property `contentType` are identical in both options. The first three or four `Accessor` objects belong to the cutting plane, and the last three or four `Accessor` objects belong to the actual shape.

Moreover, accessors of cutting planes can be simplified. The only difference between them appears in the `min` and `max` attributes of their `POSITION` (second) accessor: Accessors of the first 19 cutting planes (numbered from `P01` to `P19`) and the least 12 planes (numbered from `P20` to `P31`) are identical. This feature is derived from the size of the cutting planes: `P01-P19` result in the same dimensions on all global axes, while `P20-P31` do not satisfy this criterion and result in asymmetric sizes. As a result, only four `Accessor` objects need to be stored: we chose planes `P01`, `P10`, `P16`, and `P20`. Furthermore, these accessors can be stored in the global configuration and applied to all scenarios permuted from any 3D shape.

Since the number of possible permutations results from the number of scaled shapes, the number of rotation vectors and the number of intersecting planes, a total number of $32 * 4$ accessors should be stored to yield the accessors of each permutation of a shape. On the other hand, only the property `count` of each `Accessor` object should be kept with the `min` and `max` properties of the second accessors individually; other properties have globally identical values.

Properties `bufferViews` and `buffers`

The values of these properties can be computed from the properties of `Accessor` objects, using their order and properties `type`, `contentType`, and `length`.

Data chunk

As can be derived from the description of the JSON chunk, binary data are self-stored in each asset, and `Accessor` and `BufferView` objects can be used to retrieve a required sub-sequence of the binary encoded data. Now we can ex-

amine whether finding a pattern in these byte sequences is possible, as we know that most JSON properties — such as the `buffers`, `bufferViews`, and `accessors` — can be clearly separated into their shape-dependent and plane-dependent components.

The sequence of `Accessor` objects tells the answer. The first part of each binary chunk belongs to the cutting plane, while the rest contains the binary data of the actual mesh. Thus, the first part of the binary data — like the properties of the JSON document — can be removed from individual assets and stored in a global document. The first 1984 bytes of each buffer belong to the cutting planes; the rest of the bytes belonging to the actual shape in the case of textured assets. Otherwise, the first 1536 bytes belong to the cutting plane.

4.3. Reduced storage of the dataset for efficient use in application

In the previous sections, we described the basic features of scenarios. Based on the background knowledge, various data formats can be designed to store the local and global features of assets. In this section, we introduce the multi-level scheme allowing us to reduce the needed storage to store the dataset and encode the information in the most suitable format for the actual use case.

Figure 4.8 shows that the encoding and decoding steps form a pipeline of individual operations on the dataset; the input of each encoding function is the output of the upper layer, and the input of each decoding function is the output of the lower layer. Each level has its encoding and decoding functions, denoted with $f_E^{\circ L}(D^{L-1})$ and $f_D^{\circ L}(D^L)$, where subscripts E and D refer to *Encoding* and *Decoding*, while superscript $\circ L$ refers to the number of the corresponding level (where $\circ L = 0$ denotes the original dataset and levels are numbered from 1 to 4 inclusive). Thus, the original assets and encoded documents are noted with D_G^L, where the optional subscript G notes the ID of the corresponding group. Moreover, documents can be textured or non-textured. Thus, additional letters T and N can be added optionally to the superscripts to distinguish them, resulting in notations $D_G^{N\circ L}$ and $D_G^{T\circ L}$.

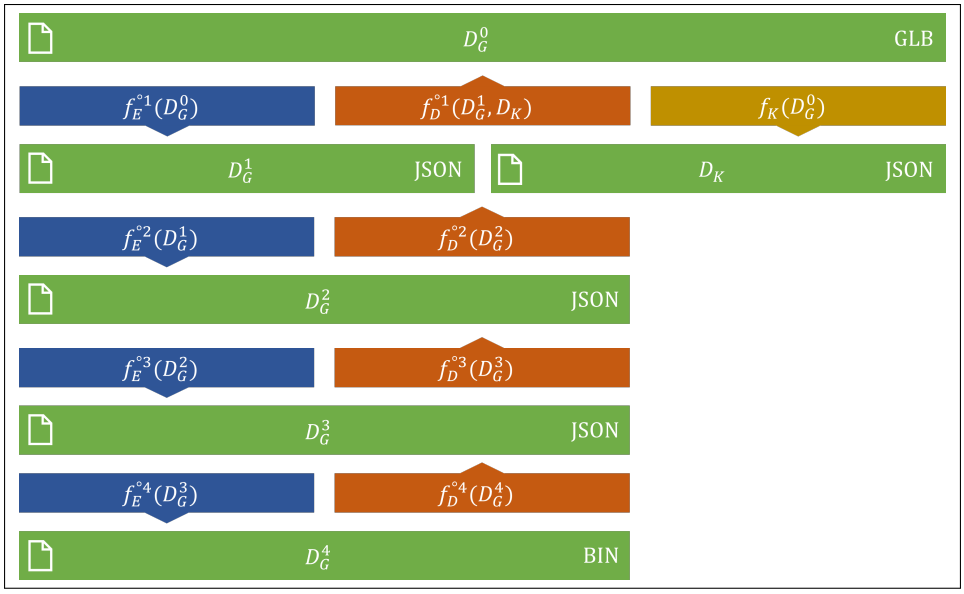


Figure 4.8: The multi-level encoding scheme.

4.3.1. Key document

Introduce function $f_K(D_G^0)$ that extracts the globally identical features from a given dataset, including each property mentioned above, and then encodes them in a JSON document.

The function can be called with any group of assets since the retrieved properties are globally identical. Consequently, this function must be called only once on a dataset, allowing us to retrieve the needed values in each invocation of $f_D^{\circ 1}(D_G^1)$. Thus, the parameter list of the first decoding function must be extended by introducing another parameter D_k, resulting in the form $f_D^{\circ 1}(D_G^1, D_K)$. The size of the document is 22,958 bytes in the case of textured documents and 19,000 bytes in the case of non-textured documents, using no indentation nor spaces between tokens. On the other hand, the structure of this document can be optimized to achieve a smaller size. However, multiple values should be fetched during the decoding of an asset. Thus, we prefer efficient accessibility to the minimized size of a single document. In this approach, needed values can be easily fetched after reading the document into memory; then, only accessing entries from the dictionary is needed instead of applying transforms on any value.

4.3.2. First level

The encoding function $f_E^{\circ 1}(D_G^0)$ consumes a set of original assets that belong to the same group and then returns a JSON document denoted with D_G^1, containing all the properties that should be stored to reconstruct the original assets with the use of D_k. As Figure 4.9 suggests, the content of the binary chunk is stored without any processing, so the content of the buffer sequence is encoded as a single series of characters, following their original order. The hexadecimal representation of the bytes was chosen since the numerical types stored in the binary chunk have sizes of 2 or 4 bytes. Thus, an encoding is needed such that each value can be encoded with a different series of characters without padding.

The decoding function $f_D^{\circ 1}(D_G^1, D_k)$ reconstructs the original assets of group G by combining the relevant local and global values stored in the properties of D_G^1 and D_k.

With the use of $f_E^{\circ 1}(D_G^0)$, storage sizes can be reduced with 99.91% in the case of $D_G^{T\circ 1}$, achieving a compression ratio 1,065 compared to $D_G^{T\circ 0}$ dataset. Both compression ratio and saved storage size are similar in the case of $D_G^{N\circ 0}$ and $D_G^{N\circ 1}$. Table 4.1 contains more details about the analysis.

```
{
    "accessors": {
        "0100": {
            "000": [24, 24, 24, 30 ],
            "001": [24, 24, 24, 30 ],
            ...
        },
        "0101": {...},
        ...
    },
    "accessors-max": {
        "0100": [1, 0.9999990463256836, 1],
        "0101": [1, 1, 1],
        ...
    },
    "accessors-min": {
        "0100": [-1, -0.9999990463256836, -1],
        "0101": [-1, -1, -1],
        ...
    },
    "data": {
        "0100": {
            "000": "000080bff0ff7...0c001700",
            "001": "000080bf28333...0c001700",
            ...
        },
        ...
    }
}
```


Figure 4.9: The structure of $D_1^{T\circ 01}$, with four numbers in each array of the property `accessors`. Precision errors can be also detected in the scalars of properties `accessors-max` and `accessors-min`.

Group	Textured dataset			Non-textured dataset		
	Size (bytes)	Ratio	Save (%)	Size (bytes)	Ratio	Save (%)
#01	165,618	1,619	99.94	127,728	1,636	99.94
#03	114,971	2,169	99.95	88,361	2,200	99.95
#05	218,916	1,127	99.91	168,028	1,141	99.91
#06	130,210	1,643	99.94	100,474	1,660	99.94
#07	418,206	867	99.88	321,292	876	99.89
#09	134,966	1,907	99.95	103,828	1,933	99.95
#10	361,400	943	99.89	277,286	954	99.90
#11	361,894	1,000	99.90	277,951	1,011	99.90
#12	473,861	808	99.88	363,870	816	99.88
#13	313,955	903	99.89	240,644	914	99.89
#14	320,616	1,016	99.90	245,942	1,028	99.90
#15	375,432	926	99.89	288,694	936	99.89
#16	218,048	1,319	99.92	168,302	1,331	99.92
#17	460,797	821	99.88	353,788	829	99.88
#18	219,616	1,312	99.92	169,310	1,326	99.92
#19	456,190	829	99.88	349,900	839	99.88
#20	343,280	978	99.90	263,502	991	99.90
#21	335,844	990	99.90	258,162	1,001	99.90
#22	234,426	1,253	99.92	180,552	1,267	99.92
#23	227,292	1,281	99.92	175,210	1,295	99.92
#24	239,992	1,233	99.92	184,214	1,248	99.92
Total	6,125,530	1,065	99.91	4,707,038	1,077	99.91

Table 4.1: Evaluation of $f_E^{\circ 1}$ on the textured and non-textured datasets, using the *Compression ratio* and *Saved disk space* metrics.

4.3.3. Second level

The encoding function $f_E^{\circ 2}(D_G^1)$ returns a second-level encoded JSON document D_G^2. The new document keeps properties `accessors-max` and `accessors-min` of D_G^1 but eliminates property `accessors` by processing and transforming the hexadecimal of the property `data`.

The idea of the transform operation is to encode each buffer content separately instead of encoding the original continuous sequence of bytes as a single hexadecimal value. As a result, the sliced values of property `data` in D_G^2 contain information about their lengths, as each original string literal is replaced with a list of string literals. Each list contains four values in the case of $D_G^{T\circ 2}$ and three in the case of $D_G^{N\circ 2}$. Moreover, observing property `data` of multiple D_G^2 documents, we can recognize that several subsequences appear multiple times in the lists. This feature is derived from the type of data stored in the GLB documents:

- Values of the first and second buffers are of types `VEC3` and `componentType` 5126, which combination means a 3-length array of IEEE 754 float values. Figure 4.4 shows that data of these buffers belong to `POSITION` and `NORMAL` attributes of a primitive; thus, crucial geometric information can be fetched from these series of sequences. Our meshes have similar geometric features in many cases but have common vertices and edges. Thus, the appearance of the same vector multiple times is obvious.

- Values of the third buffer (in the case of $D_G^{T\circ 2}$) contain data of textures. The type of the buffer is `VEC2`, and `componentType` is 5126, which combination means a 2-length array of IEEE 754 float values. If we decide to keep and encode the textures, these vectors contain repeated values, similar to the first two buffers.

- Values of the last buffer are scalar, 2-byte unsigned integer values, but repeated values can be detected again.

Thus, the size of D_G^2 can be decreased by collecting the set of unique scalar values in each buffer with lengths of $[8,8,8,4]$ characters in the case of $D_G^{T\circ 2}$ and $[8,8,4]$ in the case of $D_G^{N\circ 2}$.

The set of unique character sequences can be used as the keyset of a codebook, which allows us to eliminate the original sequences from the documents, replacing them with their identifiers. We introduce hash function $h_n^{\circ 2}(x)$, which returns the index of a scalar value in the sequence of the unique values fetched from the nth buffer B_n. The required compression can be performed by substituting the $s \in B_n$ representation of each scalar value with its corresponding $h_n^{\circ 2}(s)$ hash value. The keynote behind this operation is that the count of distinct scalar values (and thus each index value) requires fewer digits to be encoded than the original scalar values with a representation that is 4 or 8 characters long. Thus, the use of $h_n^{\circ 2}(x)$ decreases the size of each buffer using 3, 4, 3, and 1 bytes to hash 242, 13,713, 63, and 118 different values of the buffers using their indices. Another possible option could be to collect all the unique scalar values globally and store them in D_k. However, this approach would require $[563, 8082, 102, 118]$ different indices for the buffers. Furthermore, in the case of `B0`, it would require changing the length of the hash values from two digits to four digits. In the case of our dataset, it is better to use shorter hash values and create a codebook for each group. However, another dataset containing shapes with various vertices would require more digits in the hash values. In that case, the global approach would be more efficient.

Table 4.2 shows how many different scalar values appear in the hexadecimal representations of each buffer and how many unique values can be detected among them.

- In `B0`, `G13` has the greatest amount of unique values. One byte is enough to store 242 entries in a list and index them, ensuring 256 possible values. Thus, $h_0^{\circ 2}(x)$ returns indices that are represented with two hexadecimal digits, eliminating 6 digits (3 bytes) from each scalar in the original sequences.

- In `B1`, `G13` has the greatest amount of unique values. Two bytes are required to store 13,713 entries in a list and index them, ensuring 65,536 possible indices. Thus, $h_1^{\circ 2}(x)$ returns indices that are represented with four hexadecimal digits, eliminating 4 digits (2 bytes) from each scalar in the original sequences.

- In `B2`, `G13` has the greatest amount of unique values. One byte is enough to store 63 entries in a list and index them, ensuring 256 possible indices.

Group	B0		B1		B2		B3	
	Total	Unique (%)	Total	Unique (%)	Total	Unique (%)	Total	Unique (%)
#01	7,065	0.3	7,065	1.7	4,710	0.2	3,066	1.5
#03	4,950	0.7	4,950	7.7	3,300	0.2	1,680	1.4
#05	9,510	0.3	9,510	4.4	6,340	0.3	3,486	1.9
#06	5,544	0.4	5,544	2.9	3,696	0.2	2,436	1.6
#07	18,132	0.2	18,132	3.1	12,088	0.2	7,140	1.3
#09	5,799	1.3	5,799	8.8	3,866	0.3	2,100	2.3
#10	15,732	0.3	15,732	4.5	10,488	0.2	5,796	1.5
#11	15,696	0.2	15,696	1.9	10,464	0.1	6,048	1.3
#12	20,580	0.3	20,580	3.2	13,720	0.1	8,022	1.5
#13	13,713	1.8	13,713	9.8	9,142	0.7	4,704	2.3
#14	13,962	0.2	13,962	3.7	9,308	0.2	5,040	1.4
#15	16,224	0.4	16,224	4.7	10,816	0.1	6,552	1.4
#16	9,288	0.1	9,288	0.8	6,192	0.2	4,326	1.2
#17	20,022	0.2	20,022	2.0	13,348	0.2	7,728	1.4
#18	9,393	0.2	9,393	1.1	6,262	0.2	4,158	1.2
#19	19,890	0.2	19,890	3.7	13,260	0.2	7,140	1.4
#20	14,919	0.4	14,919	4.3	9,946	0.1	5,460	1.6
#21	14,526	0.2	14,526	1.9	9,684	0.1	5,754	1.3
#22	10,062	0.3	10,062	3.8	6,708	0.2	4,284	1.4
#23	9,726	0.4	9,726	4.3	6,484	0.2	4,284	1.3
#24	10,419	0.3	10,419	3.8	6,946	0.2	3,780	1.4

Table 4.2: The ratio of unique scalar values in the buffers.

Thus, $h_2^{\circ 2}(x)$ returns indices that are represented with two hexadecimal digits, eliminating 6 digits (3 bytes) from each scalar in the original sequences.

- In `B3`, `G12` has the greatest amount of unique values. One byte is enough to store 118 entries in a list and index them, ensuring 256 possible indices. Thus, $h_3^{\circ 2}(x)$ returns indices that are represented with two hexadecimal digits, eliminating 2 digits (1 byte) from each scalar in the original sequences.

Another possible option could be to collect all the unique scalar values globally and store them in D_k. However, this approach would require $[563, 8082, 102, 118]$ different indices for the buffers. Furthermore, in the case of `B0`, it would require changing the length of the hash values from 2 digits to 4 digits. In the case of our dataset, it is better to use shorter hash values and

create its own codebook for each group. However, another dataset containing shapes with various vertices would require more digits in the hash values. In that case, the global approach would be more efficient. `G13` demonstrates this feature since meshes of this group contain the greatest number of vertices having various coordinates, which results in the largest number of unique hexadecimal sequences.

Group	Textured dataset				Non-textured dataset			
	Ratio		Save (%)		Ratio		Save (%)	
	Actual	Total	Actual	Total	Actual	Total	Actual	Total
#01	2.680	4,339	62.68	99.98	2.453	4,014	59.23	99.98
#03	2.479	5,377	59.66	99.98	2.237	4,922	55.30	99.98
#05	2.613	2,945	61.74	99.97	2.376	2,710	57.91	99.96
#06	2.628	4,318	61.96	99.98	2.399	3,982	58.32	99.97
#07	2.668	2,314	62.52	99.96	2.431	2,130	58.87	99.95
#09	2.421	4,617	58.69	99.98	2.177	4,208	54.06	99.98
#10	2.619	2,471	61.82	99.96	2.378	2,269	57.95	99.96
#11	2.709	2,708	63.08	99.96	2.474	2,501	59.58	99.96
#12	2.666	2,154	62.49	99.95	2.427	1,980	58.79	99.95
#13	2.400	2,168	58.34	99.95	2.155	1,970	53.60	99.95
#14	2.647	2,690	62.23	99.96	2.410	2,478	58.50	99.96
#15	2.600	2,407	61.54	99.96	2.358	2,207	57.59	99.95
#16	2.722	3,591	63.26	99.97	2.498	3,325	59.96	99.97
#17	2.708	2,223	63.07	99.96	2.474	2,051	59.58	99.95
#18	2.716	3,564	63.18	99.97	2.489	3,300	59.83	99.97
#19	2.652	2,198	62.29	99.95	2.413	2,024	58.55	99.95
#20	2.616	2,558	61.77	99.96	2.373	2,350	57.85	99.96
#21	2.702	2,674	62.99	99.96	2.468	2,470	59.48	99.96
#22	2.623	3,288	61.88	99.97	2.388	3,025	58.12	99.97
#23	2.600	3,332	61.53	99.97	2.363	3,059	57.69	99.97
#24	2.632	3,245	62.01	99.97	2.396	2,991	58.26	99.97
Total	2.633	2,804	62.02	99.96	2.396	2,580	58.26	99.96

Table 4.3: Evaluation of $f_E^{\circ 2}$ on the textured and non-textured datasets, using the *Compression ratio* and *Saved disk space* metrics.

The decoding function $f_D^{\circ 2}(D_G^2)$ consumes a second-level encoded document and returns the corresponding first-level encoded document (D_G^1). The function splits the hexadecimal series of hash values encoded in property `data` and replaces them with the original sequences from property `data`. As the final step, the function merges the elements of each list of property `data` and re-

adds property `accessors` to the document containing the size of each buffer.

With the use of $f_E^{\circ 2}(D_G^1)$, storage sizes can be reduced with 62.02% in the case of $D_G^{T\circ 2}$, achieving a compression ratio 2.633 compared to $D_G^{T\circ 1}$ dataset, 2,804 compared to $D_G^{T\circ 0}$. Table 4.3 contains more details about the analysis. The codebook is added to the document as property `data-cb-byte`.

4.3.4. Third level

The encoding function $f_E^{\circ 3}(D_G^2)$ consumes a second-level encoded JSON document (D_G^2) and returns a third-level encoded JSON document (D_G^3). Similarly to function $f_E^{\circ 2}(D_G^1)$, its output keeps the `accessors-max` and `accessors-min` properties of the document D_G^2 but applies a second hash function $h_n^{\circ 3}(x)$ on the values of property `data`, where subscript n denotes the index of the buffer.

The hexadecimal values still contain significant redundancy, observing the `data` properties of D_G^2 documents. The purpose of this feature is simple, as we have already hashed each scalar value, but the first three buffers contain vectors of types `VEC3`, `VEC3`, and `VEC2`. Not just the same scalar values, but the same vectors appear multiple times in the original sequences. The first and second buffers contain triplets of `float` values, while the third vector (that encodes the textures) contains `integer` pairs. Thus, the statistical features of the subsequences can be analyzed again, verifying that several triples and pairs of hash values appear multiple times.

Table 4.4 shows how many different vectors appear in the buffers of each group and how many unique values can be detected among them.

- In `B0`, `G13` has the greatest amount of unique values. Two bytes are required to store 789 entries in a list and index them, ensuring 65,536 possible indices. Thus, $h_0^{\circ 3}(x)$ returns indices that are represented with 4 hexadecimal digits, eliminating 4 digits (2 bytes) from each vector's sequence.

- In `B1`, `G13` has the greatest amount of unique values. Two bytes are required to store 1,179 entries in a list and index them, ensuring 65,536 possible indices. Thus, $h_1^{\circ 3}(x)$ returns indices that are represented with four hexadecimal digits, eliminating 8 digits (4 bytes) from each vector's sequence.

Group	B0			B1			B2		
	Total	Unique	(%)	Total	Unique	(%)	Total	Unique	(%)
#01	2,355	165	7.0	2,355	93	3.9	2,355	16	0.7
#03	1,650	159	9.6	1,650	316	19.2	1,650	6	0.4
#05	3,170	137	4.3	3,170	265	8.4	3,170	18	0.6
#06	1,848	172	9.3	1,848	98	5.3	1,848	11	0.6
#07	6,044	407	6.7	6,044	480	7.9	6,044	24	0.4
#09	1,933	217	11.2	1,933	339	17.5	1,933	23	1.2
#10	5,244	266	5.1	5,244	586	11.2	5,244	23	0.4
#11	5,232	210	4.0	5,232	277	5.3	5,232	12	0.2
#12	6,860	279	4.1	6,860	479	7.0	6,860	19	0.3
#13	4,571	789	17.3	4,571	1,179	25.8	4,571	82	1.8
#14	4,654	249	5.4	4,654	470	10.1	4,654	21	0.5
#15	5,408	592	10.9	5,408	530	9.8	5,408	14	0.3
#16	3,096	170	5.5	3,096	107	3.5	3,096	16	0.5
#17	6,674	529	7.9	6,674	453	6.8	6,674	29	0.4
#18	3,131	163	5.2	3,131	137	4.4	3,131	10	0.3
#19	6,630	205	3.1	6,630	499	7.5	6,630	28	0.4
#20	4,973	354	7.1	4,973	602	12.1	4,973	10	0.2
#21	4,842	341	7.0	4,842	365	7.5	4,842	17	0.4
#22	3,354	260	7.8	3,354	308	9.2	3,354	25	0.7
#23	3,242	217	6.7	3,242	190	5.9	3,242	17	0.5
#24	3,473	148	4.3	3,473	314	9.0	3,473	18	0.5

Table 4.4: The ratio of unique vector values in the buffers.

- In `B2`, `G13` has the greatest amount of unique values. One byte is enough to store 82 entries in a list and index them, ensuring 256 possible indices. Thus, $h_2^{\circ 3}(x)$ returns indices that are represented with two hexadecimal digits, reducing the representation of each sequence representation with 2 digits (1 byte).

- As `B3` contains scalar values, thus, $h_4^{\circ 3}(x)$ should not be used.

The decoding function $f_D^{\circ 3}(D_G^3)$ consumes a third-level encoded JSON document (D_G^3) and returns a second-level encoded JSON document (D_G^2). Similarly to function $f_D^{\circ 2}(D_G^2)$, it splits the hexadecimal sequences of property `data`, replaces the hash values with the original values from property `data-cb-byte-2`, and then joins the strings. Finally, property `data-cb-byte-2` is removed to retrieve format D_G^2.

Adding the codebook as property `data-cb-byte-2` and applying function $h_n^{\circ 3}(x)$ on property `data`, storage sizes could be reduced with 37.89% in the

Group	Textured dataset				Non-textured dataset			
	Ratio		Save (%)		Ratio		Save (%)	
	Actual	Total	Actual	Total	Actual	Total	Actual	Total
#01	1.690	7,335	40.84	99.99	1.657	6,651	39.65	99.98
#03	1.413	7,599	29.24	99.99	1.353	6,659	26.09	99.98
#05	1.640	4,828	39.02	99.98	1.598	4,332	37.43	99.98
#06	1.625	7,017	38.47	99.99	1.584	6,308	36.87	99.98
#07	1.645	3,806	39.22	99.97	1.600	3,409	37.50	99.97
#09	1.402	6,470	28.65	99.98	1.346	5,665	25.72	99.98
#10	1.597	3,945	37.36	99.97	1.547	3,511	35.36	99.97
#11	1.736	4,700	42.39	99.98	1.700	4,251	41.17	99.98
#12	1.688	3,637	40.77	99.97	1.647	3,261	39.27	99.97
#13	1.291	2,798	22.53	99.96	1.230	2,424	18.72	99.96
#14	1.621	4,360	38.30	99.98	1.574	3,901	36.47	99.97
#15	1.554	3,741	35.66	99.97	1.499	3,310	33.31	99.97
#16	1.733	6,223	42.30	99.98	1.702	5,661	41.26	99.98
#17	1.671	3,713	40.14	99.97	1.628	3,340	38.58	99.97
#18	1.724	6,144	41.99	99.98	1.690	5,576	40.82	99.98
#19	1.686	3,707	40.70	99.97	1.647	3,332	39.27	99.97
#20	1.565	4,003	36.09	99.98	1.510	3,548	33.76	99.97
#21	1.660	4,439	39.76	99.98	1.616	3,991	38.10	99.97
#22	1.588	5,221	37.02	99.98	1.541	4,662	35.12	99.98
#23	1.635	5,447	38.84	99.98	1.593	4,873	37.22	99.98
#24	1.637	5,312	38.92	99.98	1.594	4,769	37.28	99.98
Total	1.610	4,514	37.89	99.98	1.563	4,034	36.02	99.98

Table 4.5: Evaluation of $f_E^{\circ 3}$ on the textured and non-textured datasets, using the *Compression ratio* and *Saved disk space* metrics.

case of textured documents, achieving a compression ratio 1.610 compared to set of $D_G^{T\circ 2}$ dataset, 4,034 compared to $D_G^{T\circ 0}$ dataset. Table 4.5 contains more details about the analysis.

4.3.5. Fourth level

All the previous encoding functions returned valid JSON documents. JSON documents have a lightweight syntax and much less overhead than other formats, such as XML. However, the size of the dataset can be reduced by introducing a binary format instead of a text format. This approach eliminates all tokens of JSON syntax, and our identifiers, as well as the property `data`, can be encoded as a sequence of bytes instead of hexadecimal character sequences.

Moreover, floating numbers in JSON documents are encoded using multiple precision digits but can be encoded with the use of IEEE 754 types. Thus, the encoding function $f_E^{\circ 4}(D_G^3)$ encodes the properties of D_G^3 and decreases the size of the dataset using binary encoding. Figure 4.10 describes the binary format of $D_G^{T\circ 4}$ and $D_G^{N\circ 4}$ documents, while Table 4.6 contains more details about the analysis.

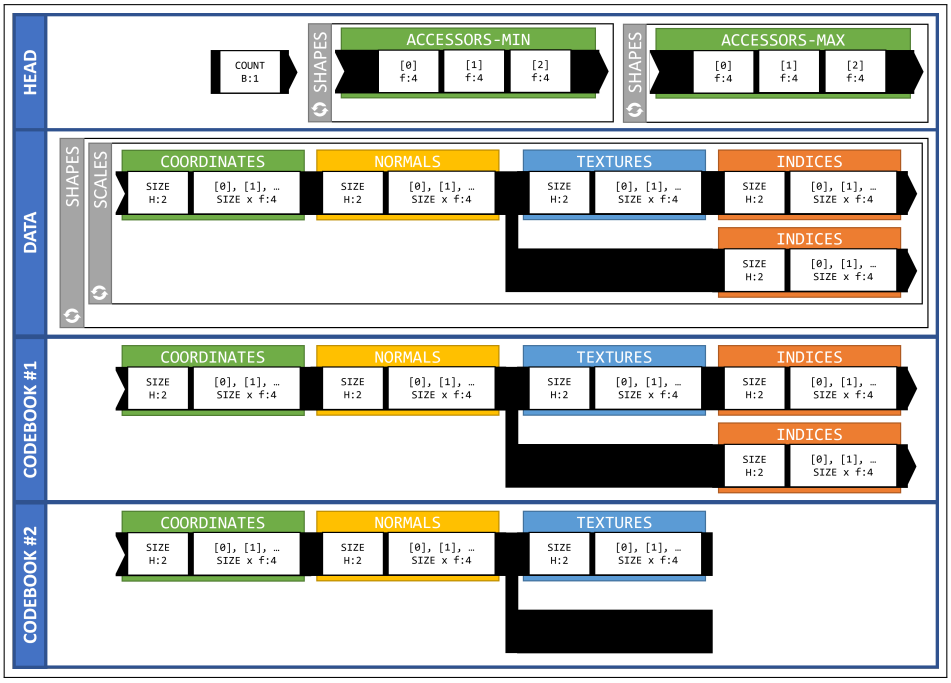


Figure 4.10: The structure of $D_G^{T\circ 4}$ and $D_G^{N\circ 4}$ binary files. Notations of types are derived from the `struct` module of Python. (Python Software Foundation, 2022)

4.4. Evaluation

4.4.1. Verification

As a first approach, it seems evident that the encoding and decoding algorithms can be verified by encoding and decoding all the assets, then comparing the original and the retrieved files bitwise. However, the bitwise equality of the original and the decoded dataset cannot be guaranteed due to the use of IEEE 754 types. As Figure 4.9 shows, the raw output of Blender already contains precision errors that can be derived primarily from the use of type IEEE 754 as a

Group	Textured dataset				Non-textured dataset			
	Ratio		Save (%)		Ratio		Save (%)	
	Actual	Total	Actual	Total	Actual	Total	Actual	Total
#01	2.099	15,394	52.35	99.99	2.116	14,071	52.74	99.99
#03	2.209	16,789	54.74	99.99	2.242	14,927	55.39	99.99
#05	2.128	10,276	53.01	99.99	2.147	9,300	53.43	99.99
#06	2.131	14,956	53.08	99.99	2.153	13,582	53.56	99.99
#07	2.118	8,062	52.79	99.99	2.136	7,280	53.18	99.99
#09	2.236	14,470	55.29	99.99	2.268	12,847	55.90	99.99
#10	2.136	8,426	53.18	99.99	2.156	7,569	53.61	99.99
#11	2.083	9,792	52.00	99.99	2.098	8,920	52.34	99.99
#12	2.103	7,650	52.46	99.99	2.120	6,912	52.83	99.99
#13	2.272	6,356	55.98	99.98	2.297	5,569	56.47	99.98
#14	2.126	9,269	52.96	99.99	2.145	8,366	53.37	99.99
#15	2.163	8,090	53.76	99.99	2.187	7,239	54.28	99.99
#16	2.072	12,897	51.75	99.99	2.085	11,801	52.03	99.99
#17	2.105	7,817	52.50	99.99	2.121	7,083	52.85	99.99
#18	2.078	12,766	51.87	99.99	2.092	11,664	52.20	99.99
#19	2.120	7,859	52.83	99.99	2.138	7,125	53.23	99.99
#20	2.158	8,640	53.67	99.99	2.183	7,747	54.20	99.99
#21	2.112	9,375	52.66	99.99	2.131	8,503	53.06	99.99
#22	2.135	11,145	53.16	99.99	2.155	10,044	53.59	99.99
#23	2.128	11,589	53.00	99.99	2.148	10,466	53.44	99.99
#24	2.123	11,279	52.90	99.99	2.142	10,217	53.32	99.99
Total	2.134	9,634	53.15	99.99	2.155	8,690	53.59	99.99

Table 4.6: Evaluation of $f_E^{\circ 4}$ on the textured and non-textured datasets, using the *Compression ratio* and *Saved disk space* metrics.

sequence of rotation, and scale operations have been applied on most of the assets in Blender.

1. Each calculation with an IEEE 754 type increases the probability of a higher error in the result.

2. Moreover, the original mesh is designed manually. Thus, designers may make minor errors in setting the coordinates of vertices. Consequently, minor errors occur in the bytes of the data chunk and the values of the JSON chunk.

3. Finally, our decoding algorithm applies a multiplication on the `min` and `max` properties of `Accessor` objects to simulate the scaling operation.

If an error occurs in the data chunk, the equality of the file sizes can be guaranteed. However, if a value of the JSON chunk is represented with a precision error, the file size changes. Thus, a comparison method should be implemented, and in using it, the verification can be performed:

1. The properties of their JSON chunks should be compared recursively. In the case of floating values, their difference should be above a given threshold ε. Objects must contain the same key-value pairs, while the order of the elements in two arrays should also be the same.

2. The binary chunks can be compared bitwise, except the sequences that belong to a buffer using floating values. In that case, the comparison must be performed using the given threshold ε.

3. Only globally unique metadata (such as property `asset`) can be reconstructed and checked since the algorithm does not code any metadata in D_G^1 but in D_k.

The comparison can be formed on the dataset using $\varepsilon = 0.00000005$, which is an acceptable value for type IEEE 754.

4.4.2. Analysis

We have already calculated the *compression ratio* and *saved disk space* of each encoding level in the description of each encoding function using the textured and non-textured versions of the dataset. However, the runtime of every function should be measured to determine whether the method is acceptable in practice. The aim of the compression is to decrease the size of the dataset, making the storage in file systems and databases possible or easier. Thus, original assets should be decoded without requiring time-consuming calculations, enabling applications to serve content online. Table 4.7 shows the time complexity of each decoding function, including all the previous decoding functions (that need to be executed) and I/O operations.

During the analysis, we found that the features of the encoding scheme offer an alternate process for yielding the dataset from Blender, decreasing the runtime significantly:

1. Export only the subset of assets from Blender, which is required as an input of $f_K(D_G^0)$ and $f_E^{\circ 1}(D_G^0)$. Denote the set of assets with D_{G*}^0.
2. Create D_k with function call $f_K(D_{GS}^0)$.
3. Encode each group to retrieve D_G^1 with a function call $f_E^{\circ 1}(D_{G*}^0)$.
4. Decode each document to retrieve the full dataset with a function call $f_D^{\circ 1}(D_G^1, D_k)$.

As Table 4.8 shows, the original exporting process of Blender requires an average of 33,886 s. On the other hand, the exporting process of the subset requires only an average of 478.1459 s; the encoding process needs 0.2605 s. Finally, the decoding process needs an average of 498.7475 s. Thus, the alternate process requires an average of 977.1520 s instead of Blender's 33,886 s, resulting in 2.8836% of the original runtime by eliminating geometric computations and permuting the assets without using Blender.

Group	$D_G^{T\circ 1}$	$D_G^{T\circ 2}$	$D_G^{T\circ 3}$	$D_G^{T\circ 4}$	$D_G^{N\circ 1}$	$D_G^{N\circ 2}$	$D_G^{N\circ 3}$	$D_G^{N\circ 4}$
01	23.89	24.38	23.98	24.47	22.89	22.71	23.48	23.88
03	24.08	23.90	23.75	24.43	22.99	22.97	23.42	23.51
05	19.38	19.13	19.64	19.27	18.67	18.38	19.18	18.75
06	19.52	19.50	19.90	19.17	18.66	18.50	19.30	19.04
07	24.35	24.65	25.39	24.27	22.70	23.34	23.67	23.67
09	24.29	24.87	24.65	24.80	23.23	23.18	23.68	23.84
10	24.88	24.93	24.63	24.26	23.53	23.01	24.04	24.09
11	27.13	26.86	27.61	26.35	25.76	26.15	26.38	26.49
12	24.28	24.83	24.43	24.46	23.58	23.47	23.92	23.96
13	19.40	20.17	19.76	19.53	18.53	18.73	19.26	19.08
14	24.21	24.64	25.35	24.47	23.23	23.48	24.13	23.91
15	24.30	24.99	25.28	24.45	23.25	23.93	23.60	24.29
16	23.94	24.90	24.69	24.27	23.35	23.83	23.79	23.84
17	24.45	25.05	25.21	24.17	23.25	23.80	24.14	24.18
18	24.29	24.66	24.94	24.32	23.13	23.38	23.78	23.93
19	24.70	25.01	24.92	24.03	23.09	23.91	24.21	23.96
20	24.43	25.47	25.28	24.48	23.10	24.14	24.03	23.77
21	24.48	24.74	25.12	24.60	23.38	23.81	23.92	23.87
22	24.29	24.83	24.93	24.11	23.47	23.52	23.50	23.57
23	24.08	24.82	24.97	24.09	23.23	24.14	24.10	24.33
24	24.37	25.06	24.56	24.43	23.18	23.91	23.94	23.74
Total	498.75	507.38	508.98	498.42	476.20	482.30	489.46	489.71

Table 4.7: Decoding time in seconds of each function.

Group	Original (s)	Enhanced (s)				Ratio (%)
		Exporting	Encoding	Decoding	Sum	
01	1,624.3172	22.7412	0.0158	23.8874	46.6444	2.8716
03	1,636.5026	23.3132	0.0162	24.0778	47.4072	2.8969
05	1,307.6983	19.1244	0.0082	19.3816	38.5143	2.9452
06	1,300.7097	18.7861	0.0112	19.5166	38.3140	2.9456
07	1,633.5453	23.3287	0.0124	24.3548	47.6960	2.9198
09	1,632.9955	23.1342	0.0125	24.2930	47.4398	2.9051
10	1,671.4794	23.1511	0.0083	24.8838	48.0431	2.8743
11	1,839.2504	25.4292	0.0156	27.1265	52.5714	2.8583
12	1,638.0409	23.4728	0.0121	24.2773	47.7622	2.9158
13	1,321.5149	19.1439	0.0062	19.4022	38.5523	2.9173
14	1,633.9655	23.8631	0.0125	24.2062	48.0819	2.9426
15	1,654.7123	23.3419	0.0156	24.3038	47.6613	2.8803
16	1,657.0160	23.2236	0.0094	23.9401	47.1731	2.8469
17	1,672.3261	23.3899	0.0156	24.4518	47.8574	2.8617
18	1,656.3157	23.4130	0.0110	24.2897	47.7137	2.8807
19	1,686.8150	23.1013	0.0156	24.7023	47.8192	2.8349
20	1,680.0384	23.1084	0.0094	24.4292	47.5470	2.8301
21	1,684.0271	23.4228	0.0156	24.4811	47.9195	2.8455
22	1,651.6237	23.2053	0.0183	24.2945	47.5181	2.8771
23	1,649.9467	23.2368	0.0094	24.0782	47.3244	2.8682
24	1,654.0871	23.2149	0.0094	24.3676	47.5919	2.8772
Total	33,886.9280	478.1459	0.2605	498.7456	977.1520	2.8836

Table 4.8: Time complexities of the original (Blender) and enhanced (hybrid) exporting processes.

4.5. Remarks

1. Each calculation was performed on a Zenbook UX433FA-A5082T notebook with an SSD and OS Windows 11.

2. During the measurements, only our Blender script or standalone Python scripts were executed on the computer. All the other non-essential processes had been stopped, including *Windows Defender*.

3. The Blender script was executed using the built-in interpreter of Blender 3.3, using our wrapper script.

4. The wrapper script and encoding process were interpreted with Python version 3.10.6 in a Miniconda 4.14.0 environment.

5. Each mentioned runtime is an average of processes in the case of our Blender script, and five processes in the case of our encoding and decoding functions.

6. A pre-processing step was executed before the encoding process to guarantee that all the shapes had the same materials without precision errors that affected the calculations. The material shown in Figure 4.5 has been added to all the assets in this step.

4.6. Summary

4.6.1. Results

This chapter introduced a lossless encoding scheme designed to represent the GLB assets of MCT scenarios efficiently. The widely-used GLB format self-encodes the geometrical information with the represented shapes' metadata. Examining the structure of the GLB documents, we retrieved all the essential pieces of information to construct the documents, eliminating redundancy from the dataset. Information about scenarios can be represented with a global document and one document for each scenario group. The encoding scheme has four layers, each depending on the previous one. The decoding process is efficient; therefore, scenarios can be served in real-time. Moreover, we proved that yielding only a subset of scenarios from Blender and reconstructing the whole dataset is more efficient than the original generation process.

4.6.2. Availability

Our standalone Python package for encoding GLB assets can be found in our GitHub repository: `https://github.com/viskillz/viskillz-glb`. The source code is published under Apache-2.0 license.

4.6.3. Relevant thesis

The result of this chapter was published in MDPI's *Education Sciences* journal. The journal had an impact factor of 3.0 and was ranked as Q2 in 2022.

Thesis 4

We introduced an encoding scheme consisting of four stages to handle the dataset by significantly reducing the storage space. The proposed method encodes a subset of assets from which it can decode the whole dataset with 3% time complexity compared to classical Blender's computations, exceeding the compression ratio of 10,000 and storage space saving 99.99%.

Journal paper: R. Tóth, Miklós Hoffmann, and Zichar, 2023a

5. Using the MCT dataset

5.1. Introduction

In the last section of this dissertation, we would like to summarize the development of our applications since they had an essential role in our research, achieving our main scientific results.

5.2. Application *viSkillz Play*

The development of the Android application *viSkillz Play* was started in 2019, and the alpha version of it was finished in 2020. The development of this application was essential to realize the demand for a script-aided method for generating MCT scenarios.

5.2.1. The platform

The application was a part of our project with the goal of providing methods and solutions in order to develop the spatial skills of different types of students. Our vision was to design and offer an implementation that can support individual practice and be a part of a modern classroom. Thus, the solutions focus on two main types of users: the students, who are willing to enhance their spatial skills, and instructors, who would like to improve the knowledge of their students. The architecture contains three main components, which can be divided into multiple modules.

Our goal was to develop a lightweight mobile application that consumes only the most important configuration from the web server while also trying to reduce the network traffic during the use of the application. This concept is based on two principles:

- For research purposes, the educational version of the application must have a live internet connection. This provides the opportunity to track the students' activity and guarantees that all the data is being transmitted to the researchers and the corresponding teacher or instructor.
- We would like to eliminate the unnecessary local persistence to avoid inconsistency in a large amount of data. On the other hand, several functions, such as scoreboards and missions, have various types, and new

versions can always be introduced. In the case of missions, it is strongly needed not to depend on the version and local data of the application – new types and their instances with the specified features can be added dynamically with the use of this approach, in which the main goal of the client application is only to show data that is being consumed from the API.

Finally, we do not want to use external dependencies. Thus, the application only uses a few necessary dependencies, including Google's UI dependencies. The AR function is implemented with Google's popular ARCore platform (`https://developers.google.com/ar`), which is also used in the tech company's own solutions.

5.2.2. Types of users

The application supports three different user types, as Figure 5.1 shows:

1. *Basic students* – The first group of students can access only the basic activities of the application, such as *workout*, *jogging*, *exams*, *categories* and *models*. With this set of activities, students are able to practice without any limitations, but they cannot access the gamified elements or the AR function.

2. *AR students* – Users who belong to this group can use all the activities that are accessible for the *basic students*, but they can also activate the AR function during a *workout*. With this lifeline, they can display the model of each assignment without any limitations.

3. *Gamified students* – Users of these groups can access all the activities that are visible for the *basic students*, while gamified elements are also introduced. This affects the use of AR; so-called *lifelines* are introduced. Lifelines are items that can be gained by completing various activities and can be used during workouts providing the students with some sort of help. This limits the use of the AR function.

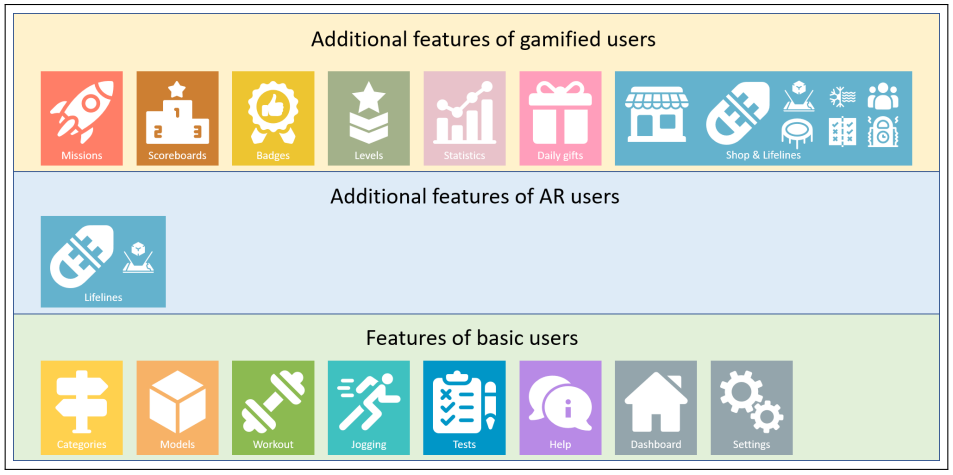


Figure 5.1: The most important features of the educational and gamification modules. Users of each group can access all the elements from the lower groups.

5.2.3. Modules

The application contains various activities which can be divided into three main groups based on their functionality:

- *Core module* – This module provides the basic functions of the application, including the authentication and authorization, the dashboard, the settings, and the help functions.
- *Educational module* – This module provides everything related to the learning process, such as the logic of different assignments and the resources organized in categories and models. The AR function also belongs to this module.
- *Gamification module* – This module contains the implementation of gamified elements, such as missions, scoreboards, experience levels, badges, lifelines, shops, daily gifts, and profiles.

5.2.4. The educational module

The most important pier of each learning environment is the set of learning materials. On the other hand, the way of organizing a large set of data in a form that provides easy access to the students and easy administration to the

instructors is not trivial. Our first exercises are derived from classic MCT exercises, while our long-term goal is to offer various types of exercises that can improve the spatial skills of students studying mathematics in secondary schools or participating in a STEM-related degree program. Thus, our goal was to create a flexible system that is easily extendable and can be reused in other applications as well.

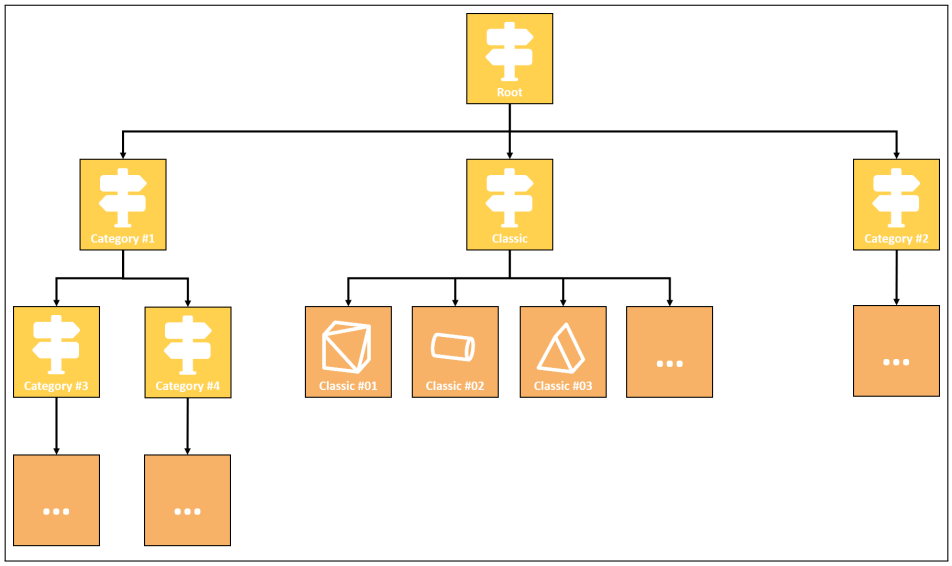


Figure 5.2: The hierarchic schema of learning resources in which models appear as leaves.

Learning resources are organized in a tree (see Figure 5.2), which contains two types of elements: categories and models. A *model* – in the case of MCT – belongs to a 3D shape that is being intersected with various planes. A model contains several exercises – whose number is equal to the number of useful cutting plane and rotation combinations – from which assignments can be generated. A *category* can both contain subcategories and models. The interface of the application hardly depends on the categories: both the navigation and the logic of assignment generation follow the hierarchy of categories.

AR function

It is one of the most important functions that are offered by our applications. As we are dealing with the development and measurement of spatial skills, it is worth considering how we can provide an easy-to-use and simple solution

to show the models to our students and give them a real chance to understand the shape of a possible answer. It is clear that students will not learn only from red and green UI elements that give brief information about their success and the correct answer.

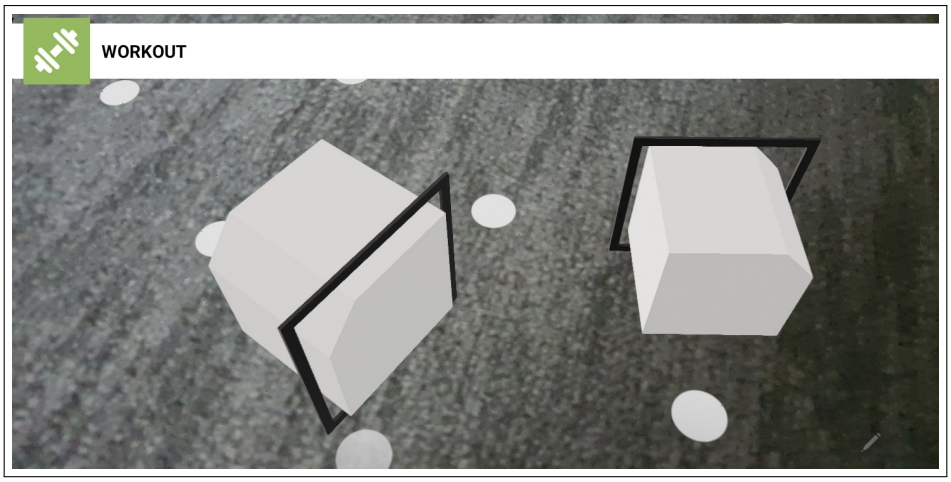


Figure 5.3: Two instances of the same shape, placed on the common surface. Circles show that a horizontal plane is detected, and the user can place one or more models on the surface.

Thus, we decided to develop a simple but useful AR function in which students are able to instantiate the model of the actual exercise by tapping on a selected point of any horizontal surface. A student is able to place multiple instances of the same instance and scale or rotate them, while the sensors of the modern devices also provide great support to move the camera and examine the models from different perspectives (see Figure 5.3). The function was built on the ARCore's instant placement API, which provides a great variety of parameters to specify the behavior of our 3D objects, such as determining and using the environment's light features to generate shadows. These options can be popular in most of the use cases, but we consider them as disturbing factors, and all the unnecessary behavior is disabled.

Workout

This activity provides an opportunity to practice without any limitations in a casual way; practice assignments are generated from a selected set of exercises (see Figure 5.4). A workout can be started in three different ways.

- First, the user can start a workout from the global (main) menu of the application. In that case, each exercise of the database can be selected and offered as the next practice exercise. We have to mention that the *freeze* lifeline can change this method in case of a gamified user; that opportunity will be described later.
- Second, the user can start a workout from the menu of a category. In that case, all the subcategories and their models will be involved in the selection algorithm that belongs to the subtree, which root is the actual category.
- Third, the user can start a workout with exercises of a single model. In that case, only different permutations of the same model will be generated in the next exercise. Such exercise offers a great opportunity to concentrate on a single shape and practice with its different rotations and perspectives.

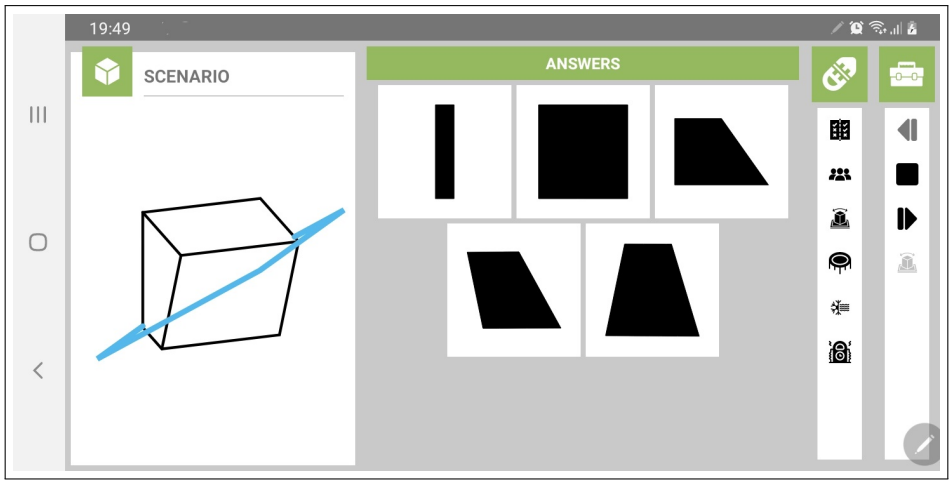


Figure 5.4: An assignment that is offered during a workout. Five possible answers are displayed to the user. On the right side of the screen, two ribbons appear. The right one provides the basic functionality, such as stopping the jogging, skipping the assignment, showing the previous one, or starting the AR function. The user belongs to the gamified group; thus, another ribbon is created, from which lifelines can be activated.

The classic MCT exercises contain five answers, from which exactly one is the correct shape of the intersection. The difficulty of an exercise strongly depends on the similarity of the offered answers. To support beginner users, we

decided to change the structure of the classic exercises. Non-gamified users can choose the difficulty of the exercises, and three, four, or five answers will be offered in the assignments based on their decision. Basically – due to our applied definition of gamification – gamification should not affect the features of the educational module; however, scoreboards, experience points, experience levels, missions, and statistics – so all the feedback elements – would be inconsistent and irrelevant to the results of the users if the same group of users does not consume exactly the same number of answers. Thus, in the case of gamified users, the number of the offered answers depends on the experience level of the user:

- 3 possible answers offered to the users who belong to experience levels 1-14.
- 4 possible answers offered to the users who belong to experience levels 15-29.
- 5 possible answers offered to the rest of the users.

The set of possible answers is deterministic for each exercise, as instructors define the possible combinations. This provides an opportunity to compare and analyze the given answers of different users. However, different lifelines are being offered to different types of users. Basic users cannot use lifelines, while AR users can activate the AR function without any limitations. For gamified users, several lifelines are offered during their workout; however, they have to be gained or purchased. Thus, we offer limited access to the extended set of lifelines. The application offers the following lifelines:

- *Augmented reality* – With the use of this lifeline, users can activate the AR function. After the activation, the AR function can be accessed until the user answers or skips the actual assignment. In the case of gamified users, the lifeline should be activated during each assignment that contains the same exercise.
- *Elimination* – With the use of this lifeline, one or more wrong answer(s) will disappear from the screen. The number of disappearing answers depends on the number of offered answers: in the case of 5 answers, 2 of them will disappear; in the case of 3 or 4 offered answers, only one

of them will disappear. Each disappearing answer is chosen randomly from the wrong ones; thus, the set of the remaining answers is non-deterministic.

- *Audience* – This lifeline is derived from the well-known "Who Wants to be a Millionaire " game in which the actual player was able to ask the audience about their preference for the answers. In our implementation, each answer of each user is aggregated that was previously given for the same exercise. This provides an easy-to-implement mechanism, and a ratio can be computed for each answer with an objective method. The user can involve the displayed data in the decision; however, it is still not guaranteed that the most popular alternative is the correct one.

- *Time machine* – This lifeline is very similar to the *audience*, but in that case, the previous answers of the actual users are being aggregated. Of course, this lifeline can only be offered if the user has answered the same exercise at least once; and it is not guaranteed that the previous answers were correct.

- *Trampoline* – With the use of this lifeline, users can skip the actual assignment without any consequence. It means that the decision to skip the exercise will not have an effect on any statistics of the user. It is important because various types of missions can deal with the accuracy of the users, such as winning streaks.

- *Freeze* – If the user does not want to meet the exercise of the actual assignment again in the next 24 hours, this lifeline offers a great opportunity to prevent the application from choosing the same exercise in the near future during the workout activity.

The assignment is evaluated immediately after the user has made his/her decision. Each assignment gets one of the following statuses:

- *Accepted* – If the user chose the correct answer.

- *Rejected* – If the user chose a wrong answer.

- *Skipped* – If the user skipped the assignment. In the case of gamified users, the use of *Freeze* lifeline does not affect the status in the database; this helps the analysis of the results.

In the case of gamified users, each correct answer is rewarded with 1 XP and 1 VisuCoin. However, different promotions – such as double score weekends – can change the number of gained XPs. Primarily, the actual assignment is shown on the screen, and users interact with the lifelines and the buttons of the possible answers. However, we would like to give quick feedback about the result of the last assignment without any annoying modals and UI elements. Thus, a button is introduced which lets the user switch back to the last assignment. The color of the button gives information about the status of the last assignment, while the user can access all the details of the last one by clicking on that button. After that, the student is able to fetch the exercise (including the answers) of the last assignment, check the selected option, and – in the case of gamified users – also fetch the data of the activated *audience* and *time machine* lifelines. Gamified users can still activate lifelines *AR* and *Freeze*, while AR users can use the AR function too.

Jogging

The UI and accessibility – it can be launched in various ways, and exercises of a model, category, or multiple categories can be offered – is very similar to the workout; however, a set of assignments is being generated instead of a single one, and users cannot activate and use any of the lifelines. Thus, users of each group meet exactly the same conditions. It offers an objective opportunity to compare the results of users from distinct groups while users still have the possibility to determine whether they want to start a new session or not. After a user starts a jogging activity, exactly 10 of the possible exercises are selected, and a list of assignments is generated. Then, the first assignment immediately shows up, and basically, a student does not leave the jogging until all the assignments are answered. Users do not get any instant feedback about the correctness of their answers during the session, and they are not able to pause or put the session behind any other application or activity. This prevents the users from cheating, like checking their previous answers for the same set of exercises. Thus, a jogging activity can be finished in three different ways:

- Basically, if the user has already answered – or skipped – each of the assignments immediately after the 10th assignment has been answered.
- If the user explicitly stops the jogging by clicking on the stop button. This

marks all the remaining assignments as skipped.

- If the `onPause()` method of the Android activity is invoked, it means that the activity is paused and no longer in the foreground because of any purpose. This event is exactly handled as in the previous case; all the remaining assignments are being marked as skipped ones.

In the background, all answers are evaluated. The set of possible statuses of an assignment is exactly the same as in the case of assignments of workout:

Due to that practice, each assignment will exactly have one status of the following ones:

- *Accepted* – If the user chose the correct answer.
- *Rejected* – If the user chose a wrong answer.
- *Skipped* – If the user skipped the assignment.

The score of an assignment can be determined with the following equation (where $i \in \mathbb{N}^+$):

$$s = \sum_{i=1}^{i=10} \begin{cases} 1, & \text{if } a_i \text{ is accepted} \\ 0, & \text{otherwise} \end{cases} \tag{1}$$

After finishing a jogging activity, the user is redirected to the UI, which contains the result of the jogging, including the resources of assignments and the chosen answer (see Figure 5.5). The feedback of a jogging activity can be accessed for up to 24 hours. To provide motivation, gamified users also claim rewards by completing a jogging activity, and various missions can be based on the use of this activity. XPs and coins are given exactly with the same method of the workout assignments, but the completion of each session is rewarded with

$$e = \max(s-7,0) \mod 5 \tag{2}$$

extra points and *VisuCoin*s, while promotions – such as double-score weekends – can also effect these values.

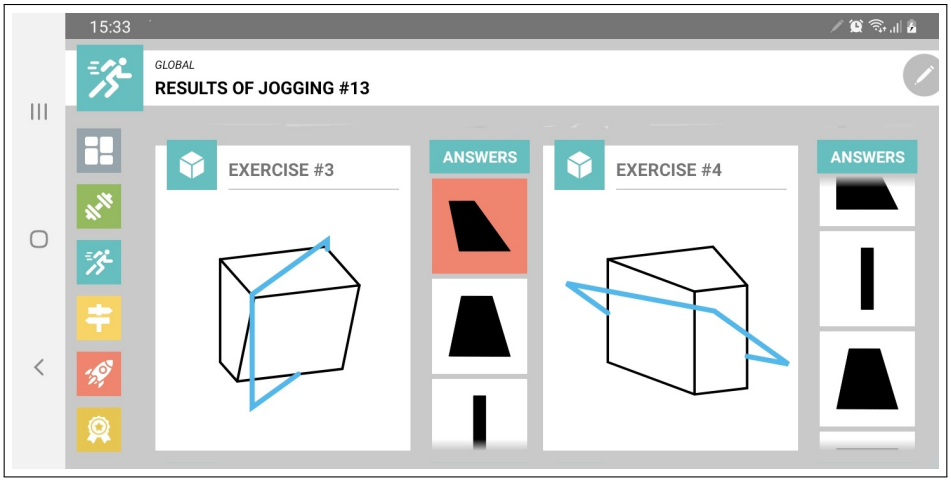


Figure 5.5: Assignments of a jogging activity. Students can display all the assignments with their corresponding answers, highlighting the selected one. Answers appear in a scrollable view, which has two columns in landscape mode. Each list of answers can be scrolled individually.

Tests

Tests — from the view of a student — are very similar to joggings, except they can be launched by the instructors. On the other hand, they are only available in the educational version, and the business logic differs more. Joggings always contain exactly 10 assignments, while tests are created by the instructors. This means that they both choose the students of a test, the assignments of a test, and the period in which the test should be solved, as well as the method of starting. Students get the keys from their instructors with the use of which they can access and start a practice session. From that point, the implementation of tests and joggings are exactly the same, except instructors can decide whether students can access their results or not, and gamified students cannot claim rewards with the use of tests.

We can determine that tests are the most important tools for an instructor or teacher who wants to check the knowledge of the students, while the method also provides a great opportunity to measure and compare the actual knowledge of users who belong to one of the control groups.

5.2.5. The gamification module

In this section, we introduce the most important elements of our gamification module. The application offers various feedback elements such as scoreboards, progress bars, statistics, badges, XPs, and XP levels. However, in this section, we highlight the scoreboards and the missions to demonstrate the mechanism of forcing the students into practicing with automatically generated missions.

Scoreboards

Scoreboards are elements that are derived from classic rank lists of games in which players are sorted based on their results, such as scores or XPs. In our scoreboards, users are sorted by the amount of XP achieved using the application. Like educational activities, scoreboards are designed over the concept of categories and models; however, XP levels will be involved too in the implementation of new boards. Thus, there are four basic types of scoreboards based on the subset of data that is being used in the generation:

- *Global scoreboard* – This scoreboard contains all gamified users of the application, comparing them by the total amount of gained XPs.
- *Category scoreboard* – A scoreboard can be constructed for each category, comparing the XPs of users gained from the corresponding subset of exercises. As in the educational features, the subset of the selected category will be used in the computations, including all the subcategories and models.
- *Model scoreboard* – As many exercises belong to the same model, various assignments can be offered based on the same mesh. Thus, a scoreboard can be generated for each model, using XPs that were gained with assignments belonging to the same group.
- *Level scoreboard* – Each experience level has a scoreboard on which the corresponding users — who are on the given level — are listed.

However, the previous types filter the set of assignments only on the type of context. Obviously, time is an important factor in gamification, especially in

educational applications. Thus, each of the scoreboards can be combined with various intervals to obtain the final set of scoreboards: activities of the actual day, week, month, and year can be selected and involved in the computation of a scoreboard. Table 5.1 contains information about whether a scoreboard is available for a given combination of interval and context.

	Global	Category	Model	Level
Total	+	+	+	+
Annual	+	+	+	-
Monthly	+	+	+	-
Weekly	+	+	+	-
Daily	+	+	-	-

Table 5.1: The possible combinations of basic scoreboard types and selected intervals. In most cases, a total, annual, monthly, weekly, and daily scoreboard can be generated based on the subset of assignments. However, only one scoreboard is being maintained for each level, and the daily interval is only being applied to the global scoreboard.

Missions can be constructed for each type of scoreboard, offering challenges and motivation, which is a possible motivation factor for our users. On the other hand, the maintenance of the scoreboards needs a high amount of computation in the case of active users and a lot of resources. For example, consider the classic 25 MCT exercises, which can serve the first wave of resources. The category of "Classic" and its 25 models result in a total of $26 \times 4 = 106$ scoreboards. Consider that we introduce only the first 30 experience levels (which has an effect on the number of given answers). A correct answer for an MCT exercise immediately triggers a change in 15 scoreboards: 5 global, 5 category, 4 model, and 1 level scoreboards are affected. Consider that 100 users perform a workout in the application, and they spend an average of 10 seconds on each assignment. This would trigger $100 \times 6 \times 15 = 90000$ update operations in a minute. Thus, scoreboards are generated only with a fixed rate of the interval, which should be determined based on the number of users, resources, and levels. However, while the front end is a lightweight application, these factors can be configured easily on the server side without any effect on the Android apps.

Missions

Missions are one of the most important elements in transforming the boring learning process into something more enjoyable. In our application, missions are built from one or more goals, and each of them asks the student to perform an action in the application. With the use of missions, we can extend the functionality of the application and try to engage and control the practice of a student. In our application, missions contain one or more goals. The student has to satisfy each of them to complete a mission. Goals can be various, and – as we previously mentioned – the introduction of a new type does not require any changes in the application. Missions can be divided into three groups:

- *Basic missions* – Missions that can be offered and completed only once. Most of the missions belong to this category.
- *Temporary missions* – Missions that are always automatically generated with the aim of providing a minor milestone that can be easily reached by spending a short time practicing. These missions show up continuously and ask the user to solve n workout assignments, start jogging or perform any other action. Constraints can also be generated on the goals, for example, mentioning a specific category or model.
- *Permanent missions* – These missions can be offered multiple times and can be completed repeatedly. For example, a mission that asks the user to solve at least 50 workout assignments on the current day belongs to this category. These missions should be defined by the administrators, while they can be offered automatically.

Each mission can have different levels of dependency, single multiple, or even none in some cases. Consider that mission A depends on mission B. In that case, mission B will be only available if mission A has been completed by the user. Thus, the sequence of missions can be precisely designed, and the practice of users can be easily supervised and controlled. The creation of most missions can also be automatized if we follow a reusable pattern in the design process. For example, based on the structure of resources, the following missions can be easily generated and used as a compass for each user:

- A mission that asks the user to submit the given percent of exercises that belong to the same model or category.

- A mission that asks the user to solve (answer correctly) the given percent of exercises that belong to the same model or category.

Of course, the relationships (dependencies) between these missions can be defined very easily. Consider mission s_i ($i \in \mathbb{N}^+$) that asks the user to submit p_i percent of the exercises, while mission s_{i+1} asks the user to submit p_{i+1} percent of the same set of exercises ($p_{i+1} > p_i$). Introduce missions c_i and c_{i+1}, which ask the user to correctly answer (solve) p_i and p_{i+1} percentage of the relevant exercises (in that order). In that scenario, s_i can be offered first to the user, while s_{i+1} and c_{i+1} are direct dependents of s_i and c_{i+1} refers to s_i as a transitive dependency (see Figure 5.6). Missions that belong to harder exercises can also depend on missions that belong to easier ones. In the case of the 25 classic MCT exercises, using $p = c = [5, 15, 25, 50, 75, 100]$, a total of 300 missions can be generated. Here are a few examples of further types of missions that can be easily generated, reused, and offered in each node of the hierarchic structure of resources:

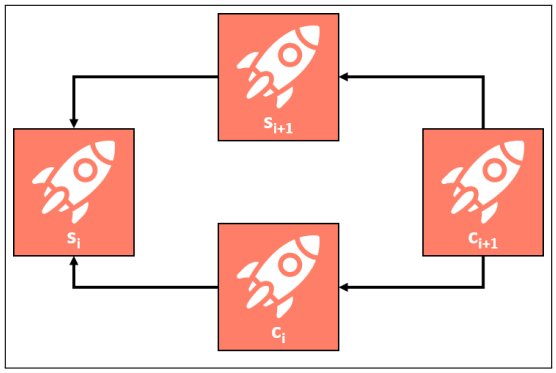


Figure 5.6: Dependencies between missions s_i, s_{i+1}, c_i and c_{i+1}. Mission c_{i+1} has a total of two direct and one transitional dependency.

1. A mission that asks the user to solve n assignments in a row without skipping an assignment or giving the wrong answer. This type of mission forces the user to practice more precisely and not choose random answers to get a chance to increment the number of XPs.

2. A mission that asks the user to solve n assignments on a given interval, which forces the user to practice with a short rush.

3. To reach position n on a specified scoreboard. This type of goal creates a much closer milestone for a user than the reach of the next level.

Also, various missions can depend on the dataset that is available in the database of the application, including the resources, behavior, and statistics of a user. With each completion of a mission, rewards are being given to users such as XPs and *VisuCoins*, while lifelines and badges can also be rewarded. Further mechanisms can be inherited from games that force the students into practice, such as double score weekends or other promotions for holidays or periods before measurements.

5.2.6. Availability

As we mentioned in the introduction, this application was our first project, and we distributed an alpha version to a small group of test users. After it, we changed the priority of the research directions due to the lack of scenarios and possibilities for personal meetings due to the COVID-19 pandemic. Therefore, we will release the application in Google Play once an adaptive exercise recommending a solution is implemented.

5.3. Application *viSkillz Browser*

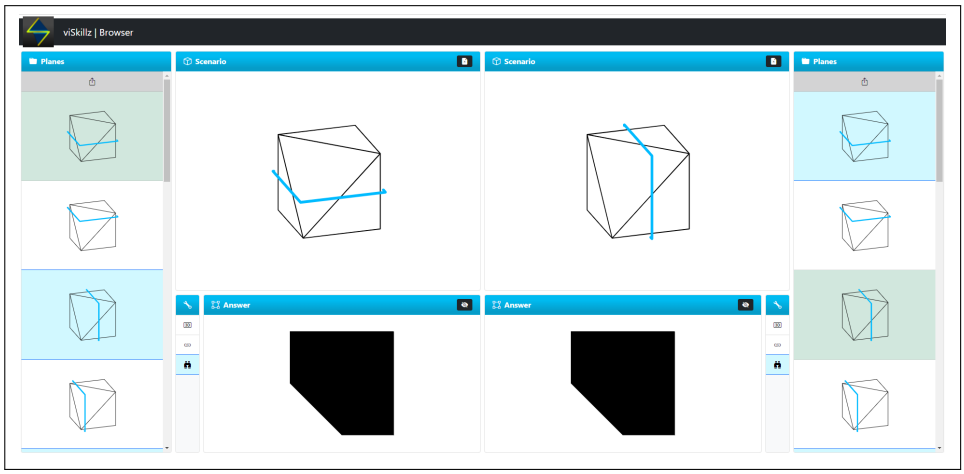


Figure 5.7: A screenshot of the *viSkillz Browser* application taken in a desktop browser.

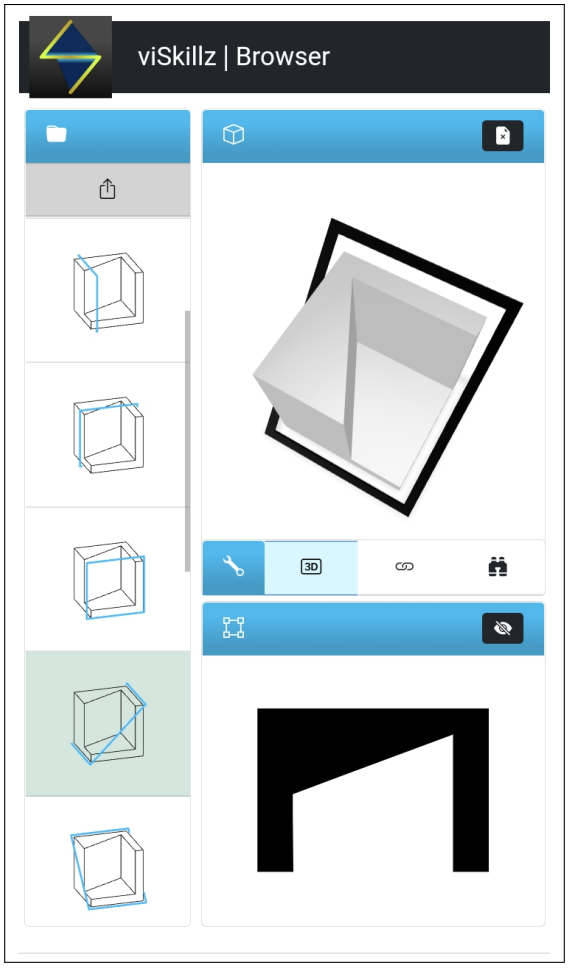


Figure 5.8: A screenshot of the *viSkillz Browser* application taken in a mobile browser, in portrait orientation.

The aim of the *viSkillz Browser* application is to provide a simple user interface with the use of which users can browse the database of permuted scenarios. The design of the user interface was derived from the well-known *Midnight Commander* and *Total Commander* applications in order to provide a tool both for instructors and their students, enabling access to all the resources in the multi-level hierarchy. The application is responsive; thus, it has different layouts on different screen sizes: on smaller screens (such as on mobile phones in portrait or landscape mode), a one-side view is being presented to the users, but on larger screens, a secondary section appears on the right side

of the window. Both sections have their own navigation panel, which makes the users able to select the scenario which they want to inspect. If a user selects a leaf node of the hierarchy of scenarios — that is, a real scenario — it is displayed in the viewer panel of the corresponding section, making the user able to check the scenario in 2D or 3D, but the shape of the intersection can also be unrevealed (see Figure 5.7 and Figure 5.8).

5.3.1. Hierarchy of scenarios

The hierarchy of the scenarios — thus, the method of navigation — is derived from the features of the permutation, resulting in the following logical structure. To access a scenario, a user should navigate through the following steps in this order:

1. **Shape** - First, the user selects a shape that is derived from the well-known sheet of MCT exercises.

2. **Orientation** - Once a shape has been chosen, the user selects its orientation, which is the combination of the camera and the rotation of the 3D mesh. The application offers two cameras that are also derived from well-known exercises.

3. **Scale** - Third, the user selects a scaling factor for the 3D shape.

4. **Plane** - Finally, a set of cutting planes are listed. Only those planes are included in the list, which does not result in empty intersections. Thus, a total number of 31 cutting planes can be displayed for each combination of *shape* and *orientation*.

The application primarily serves all the permutations (including useful and unuseful ones), but we have filtered out redundant combinations of permutation factors and some orientation, camera, and intersection combinations. Once a student selects a scenario, its 2D projection immediately appears in the viewer of the corresponding section. This lets the user check the classic form of the scenario without seeing any possible answers. If the user wants to reveal the shape of the intersection, it can be displayed by clicking on the corresponding button. Additionally, the 2D projection can be replaced by a 3D model viewer. The corresponding 3D asset is displayed in an interactive viewer

using the `model-viewer` library, offering an opportunity to rotate or scale the 3D asset both with the cursor and touch gestures. Finally, the user can check other exercises that result in the same intersection; this feature uses the output of our matching algorithm (this can be seen on Figure 5.7 using green backgrounds.

As we have already mentioned, this application is designed to support both instructors and their students or simple users who want to browse MCT scenarios. If we focus on the didactic context, the following use cases can be considered:

1. An instructor or a researcher wants to create a test that includes MCT exercises. He or she can browse the scenarios and select the ones from which a test exercise can be composed. All the resources — including both 2D and 3D assets — can be accessed and downloaded from the user interface without the need for programming skills.

2. A student can practice for a test by browsing the scenarios and trying to figure out the shape of the intersections. As a self-correction method, it is possible to show the correct answer and compare it with the predicted one. However, we expect that confused students will activate the 3D viewer at this point and figure out the differences between their initial answer and the correct shapes.

5.3.2. Availability

The application is hosted on our local virtual machine and can be accessed via URL `https://viskillz.inf.unideb.hu/browser`. The application uses the latest release of library `model-viewer` (`https://modelviewer.dev/`). Thus, check the library's reference for supported browsers and their versions.

5.4. Application *viSkillz Quiz*

As a researcher or instructor, it is not enough to prepare datasets and develop practice applications. In our short-term plans, we are going to take measurements that can verify whether the use of 3D resources can enhance the answers of the users or not. Thus, we have designed an application that merges the

functionality of surveys and practicing applications and the *viSkillz Quiz* application to offer pre-configured quizzes to its users. Each quiz can be started by entering a *token*. Multiple tokens can be assigned to a single quiz, with the use of which the access can be restricted, but additional information can also be retrieved from them, e.g., groups of users can be identified with their tokens. The application supports multiple languages (we have already introduced Hungarian, English, and Spanish), from which the user must choose one at the start of a quiz, and the list of possible languages is also determined based on the token. The main conception is to serve reusable content; thus, a quiz is a composition of multiple questions and assignments that are stored separately and can be included in more quizzes.

5.4.1. Types of entries

The quiz consists of three different types of *entries* that can be served in the chosen order:

1. *Assignment* - This entry is a classic MCT exercise (see Figure 5.9). By default, the 2D projection of the scenario and five possible shapes are being served as possible answers. As in the classic workflow, a user must choose exactly one of the shapes that describe the shape of the intersection. However, an assignment can be skipped, and the built-in 3D viewer — which is also used in the *viSkillz Browser* application — can be activated. The availability of these functions can be configured in each quiz.

2. *Instruction* - With the use of this entry, various textual contents can be displayed (see Figure 5.10). Thus, it is possible to share instructions with the users or offer them additional information about the quiz. The content of each instruction is stored and serves as a Markdown document, and the front-end application renders and displays it as an HTML document. We must mention that this solution is capable of carrying Base64-encoded images; thus, images can also be included in the instructions without real limitations. On the other hand — in some cases — it is required to format the content, especially the images used in the instructions. Thus, it is also possible to add the content of instructions as HTML and use the `style` and `class` attributes of them to format the instruction.

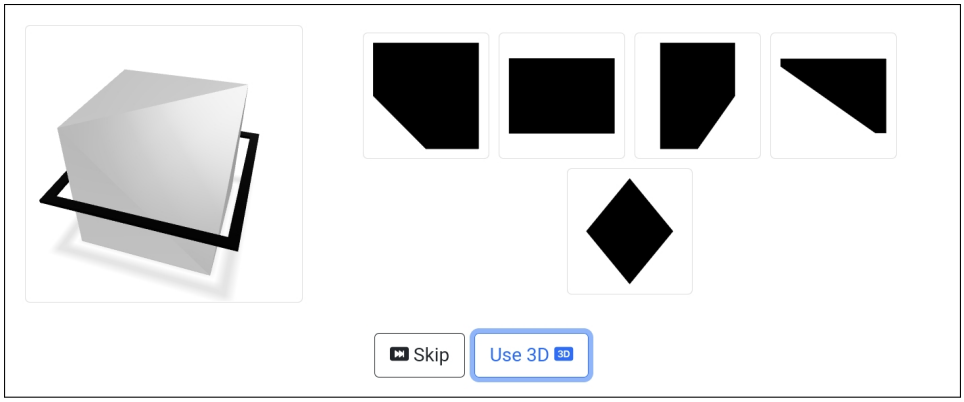


Figure 5.9: An assignment with an activated 3D viewer.

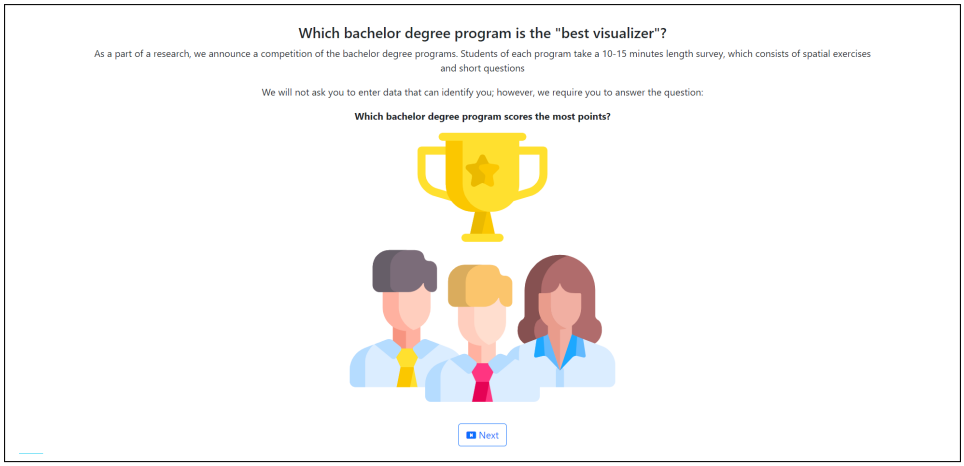


Figure 5.10: An instruction with embedded, Base64-encoded images.

3. *Question* - This entry is derived from the surveys. At this point, the application supports two types of questions, which may consist of multiple sub-types. Each question has a type of `text` or `choice` (see Figure 5.11 and Figure 5.12). Type `text` provides an opportunity to collect short answers from the users. The UI can be customized with properties: the minimum and maximum length, the corresponding form element (`input` or `textarea`), and their attributes can be specified. On the other hand, type `choice` is introduced to support the creation of `radio` and `checkbox` elements; the number of minimally and maximally required options can determine the exact type of the element.

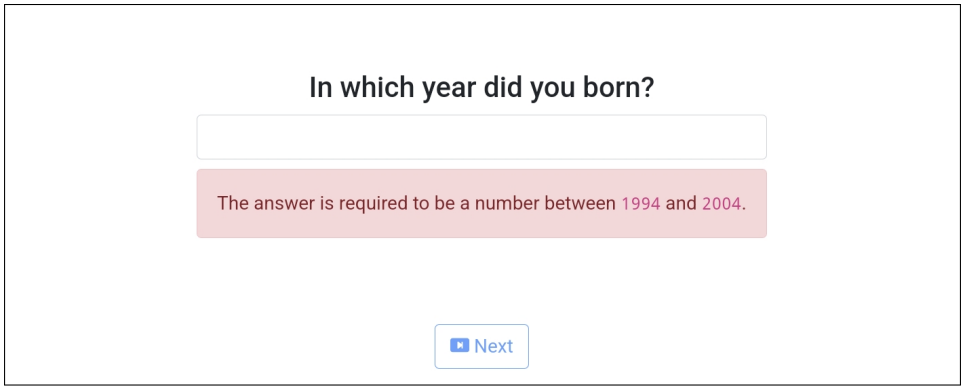


Figure 5.11: A question having `text` type, asking for the age of the user.

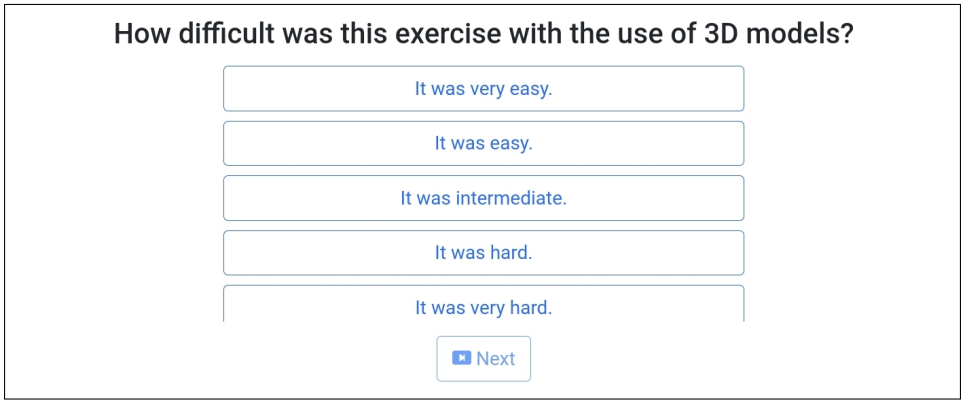


Figure 5.12: A question having `choice` type, asking for feedback regarding the last assignment that offered the 3D viewer.

Once an instructor or researcher composes a quiz, the following features can be specified:

1. The availability of the 3D viewer for each assignment.
2. Whether a question or assignment is mandatory or can be skipped.
3. A question can be displayed or omitted based on the correctness of the answer for the last assignment. This makes us able to ask more specific questions about the previous assignment, such as *Why did you skip the previous assignment?* and *How difficult was it to answer the previous assignment?*.

5.4.2. Availability

The application is hosted on our local virtual machine and can be accessed via URL `https://viskillz.inf.unideb.hu/quiz`. The application uses the latest release of library `model-viewer` (`https://modelviewer.dev/`). Thus, check the library's reference for supported browsers and their versions. A sample quiz can be started using tokens `HU` or `EN` (the first is a Hungarian version, and the second is an English version of the same quiz).

5.5. A pilot measurement

In February 2022, we organized a pilot measurement at our faculty to get feedback about our development from various students. Thus, in the first week of the spring semester, we visited lectures of our 1st and 2nd-year *Business Informatics*, *Computer Science* and *Computer Science Engineering* students and asked them to complete a 10-15 minutes long MCT measurement in the *viSkillz Quiz* application (this quiz can be started via the provided link, using the given tokens `HU` and *EN*. After a short introduction to the MCT exercises, the quiz offered 10 exercises of various difficulty levels. In the first part, each student could only use the 2D viewer of the application. In the second part of the experiment, the students got the same exercises again, but half of them could already use the 3D viewer. A total number of 263 *Computer Science*, 129 *Computer Science Engineering*, and 109 *Business Informatics* students submitted valid answers to all the exercises. From the view of this paper, the most interesting part of this measurement is the feedback from our students about the use of 2D and 3D viewers. In this quiz, a feedback question was asked after each assignment:

- In the first part — in which all students got the same, 2D versions of the assignments — the following feedback question was asked: *How difficult was this exercise?*. Students could exactly choose one out of the following options *Very easy (1)*, *Easy (2)*, *Intermediate (3)*, *Hard (4)*, and *Very hard (5)*.

- In the second part, different questions were presented for the two groups of students. The 2D group could answer the question *Was this exercise easier for the second time?*, while the 3D groups' question was *Was this*

exercise easier with the use of the 3D viewer?. In both cases, the scale was the same: *Absolutely not (1)*, *Not (2)*, *I cannot decide (3)*, *Yes (4)*, and *Absolutely yes (5)*.

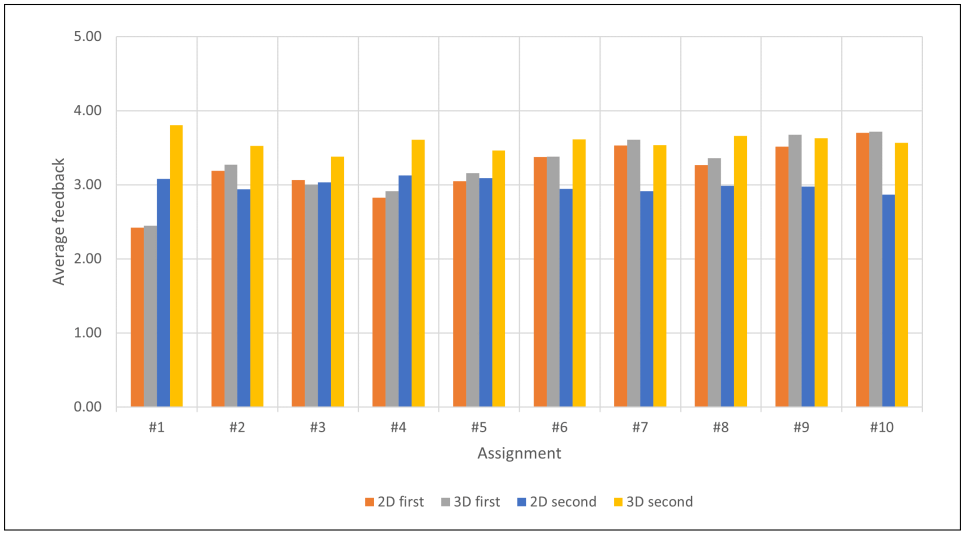


Figure 5.13: Average scores of the feedback questions.

Figure 5.13 shows the average scores of the feedback questions. It is clear that both groups gave the same answers about each assignment. However, there is a significant difference between the average scores of the second round — students with access to the 3D viewer found the assignments easier. However, Figure 5.14 shows that the use of the 3D viewer had not changed the results consistently in the second part of the quiz. There were exercises in which the students could increase their scores, but the use of the 3D viewer did not affect another assignment.

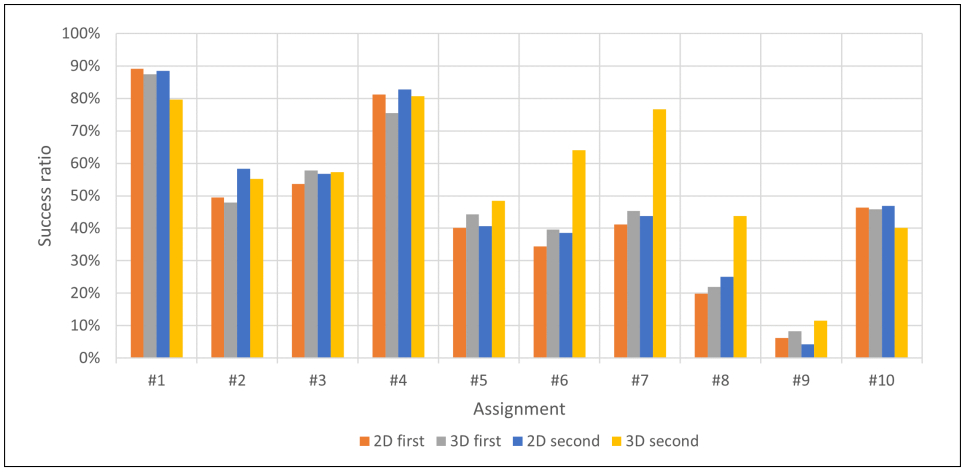


Figure 5.14: Success ratio of each assignment by parts and user groups.

Based on the measurement results, our next goal is clear: we should get more experience with the use of the 2D and 3D viewers and find the answer to the following question: *When and how does the 3D viewer change the results of an assignment?*.

5.6. Summary

5.6.1. Results

This chapter gave an overview of the different development directions of our research. First, we started to work on a gamified mobile application that offers MCT exercises to its users, enhancing the process using AR technology. Using this application, we could realize the need for the previously described dataset and the corresponding procedures. Based on our solutions, multiple applications were designed and implemented, serving MCT scenarios in real-time to their users both in 2D and 3D. Finally, we carried out a local measurement with the involvement of our students using one of the applications.

5.6.2. Relevant thesis

We gave presentations about our development directions at one Hungarian PhD conference and five international conferences, resulting in one Hungarian and three English publications in the relevant proceedings.

Thesis 5

We developed applications that serve scenarios retrieved, post-processed, and encoded using our proposed methods. The applications prove that scenarios of our dataset can be served in real-time to construct and offer MCT exercises using the available assets in 2D and 3D for various purposes.

Conference papers:

- R. Tóth, Miklós Hoffmann, Kósa, and Zichar, 2019
- R. Tóth, Zichar, and Miklós Hoffmann, 2020
- R. Tóth, Zichar, and Miklós Hoffmann, 2021
- R. Tóth, B. Tóth, Zichar, et al., 2023b

Conference talks:

- R. Tóth, Miklós Hoffmann, Kósa, and Zichar, 2020
- R. Tóth, Miklós Hoffmann, and Zichar, 2023b

6. Summary

In this dissertation, we gave an overview of our research about the design and development of procedures and applications, which aim is to enhance the development and use of Mental Cutting Test exercises. We highlighted the main contributions of this dissertation using five theses. The dissertation started with an introduction describing the background topics, such as measuring and developing spatial skills, augmented reality, and gamification. Finally, we introduced the Mental Cutting Test before clarifying our main research directions and terminology.

Our first chapter focused on designing and creating Mental Cutting Test scenarios using the popular Blender software. First, we designed 25 shapes in Blender which are derived from the well-known set of 25 classic exercises. Second, we permuted them manually, yielding 205 different shapes and forming groups. Later we designed and implemented a script-aided method capable of automatically permuting scenarios from these meshes, applying pre-defined cutting planes, rotations, and scaling operators on them. This chapter had two theses based on our journal paper published in 2021, having a rank of Q3 (R. Tóth, 2021) and another published in 2023, having a rank of Q2 and an impact factor of 3.4 (R. Tóth, B. Tóth, Miklós Hoffmann, and Zichar, 2023):

1. *We designed 205 shapes and 31 cutting planes manually in Blender. These shapes are derived from the classic Mental Cutting Test exercises and can be used to analyze the shapes and develop scripts to yield assets.*

2. *We designed a script-aided method that applies pre-defined permutation factors on the manually designed meshes and cutting planes to yield a great number of Mental Cutting Test scenarios.*

The second chapter dealt with the post-processing and analysis of the script-aided method's output. First, we proposed a method for detecting and eliminating anomalies from the intersections calculated using Blender. Second, we designed a divide-and-conquer algorithm for yielding the set of unique shapes as the possible intersections (answers) of the MCT exercises. This chapter had one thesis based on a conference paper released in 2022 (R. Tóth, B. Tóth, Zichar, et al., 2023a):

3. *We post-processed the output of the permutation script and found anomalies in the SVG assets. We designed and implemented an automated raster-based validation process based on our experience with human validation. Assets having errors due to calculation errors can be corrected, or their corresponding scenario can be dropped automatically.*

Later, we focused on the most important question about the created dataset: *How can the scenarios be stored efficiently and served in real-time for various purposes?*. As a proposed solution, we designed a multi-level encoding scheme that supports the lossless storing of the assets exceeding the compression ratio of 10,000 and the storage space saving 99.99%. Moreover, we made the generation of the assets faster, decreasing the ratio of calculations performed in Blender. The chapter had one thesis based on a journal paper released in 2023, having a rank of Q2 and an impact factor of 3.0 (R. Tóth, Miklós Hoffmann, and Zichar, 2023a):

4. *We introduced an encoding scheme consisting of four stages to handle the dataset by significantly reducing the storage space. The proposed method encodes a subset of assets from which it can decode the whole dataset with 3% time complexity compared to classical Blender's computations, exceeding the compression ratio of 10,000 and storage space saving 99.99%.*

Finally, we presented possible use cases of our main scientific results; the development of various applications served as the experience on which we realized the need for script-aided generation and lossless encoding methods, as well as we could test the designed solutions by adding them to these applications. This chapter had one thesis based on four conference papers and two conference talks (R. Tóth, Miklós Hoffmann, Kósa, and Zichar, 2019; R. Tóth, Zichar, and Miklós Hoffmann, 2020; R. Tóth, Zichar, and Miklós Hoffmann, 2021; R. Tóth, B. Tóth, Zichar, et al., 2023b; R. Tóth, Miklós Hoffmann, Kósa, and Zichar, 2020; R. Tóth, Miklós Hoffmann, and Zichar, 2023b):

5. *We developed applications that serve scenarios retrieved, post-processed, and encoded using our proposed methods. The applications prove that scenarios of our dataset can be served in real-time to construct and offer MCT exercises using the available assets in 2D and 3D for various purposes.*

7. Összefoglaló

Ebben a disszertációban átfogó képet adtunk arról a kutatásunkról, mely elsősorban a Mental Cutting Test feladatok előállítását szeretné hatékonyabbá tenni különféle eljárások és alkalmazások tervezésével és fejlesztésével. A főbb eredményeinket öt tézisben ismertettük. A disszertációt egy általános bevezetéssel kezdtük, melyben bemutattuk a kutatás hátteréül szolgáló témaköröket: a térbeli készségek mérése és fejlesztése, a kiterjesztett valóság, valamint a játékosítás egyaránt említésre került. Végül a Mental Cutting Test formátumát ismertettük, tisztázva a kutatási irányainkat és a terminológiát.

Az első fejezetben a Mental Cutting Test feladatokhoz tartozó szcenáriók tervezését és előállítását ismertettük, melyet a népszerű Blender szoftverrel valósítottunk meg. Elsőként 25 testet terveztünk a jól ismert MCT feladatsor ábrái alapján. Következő lépésként manuálisan permutáltuk az alakzatokat, 205 különböző testet előállítva és csoportokba rendezve. Később megterveztük és implementáltuk azt a szkript-alapú módszert, mely alkalmas a testekből permutált szcenáriók automatikus előállítására előre definiált metsző síkok, valamint forgatási és skálázási műveletek alkalmazásával. Ez a fejezet két tézist tartalmazott egy 2021-ben megjelent és Q3-as besorolású (R. Tóth, 2021), valamint egy 2023-ban megjelent és Q2-es besorolású, 3,4 impakt faktorú folyóiratcikk (R. Tóth, B. Tóth, Miklós Hoffmann és Zichar, 2023) alapján:

1. *205 testet és 31 metsző síkot terveztünk Blenderben. A testeket a klasszikus Mental Cutting Test feladatokból származtattuk. Az elkészített modellek elemezhetőek és a feladatok assetjeit előállítható szkript fejleszthető a segítségükkel.*

2. *Egy szkript-alapú eljárást terveztünk, mely előre definiált permutációs lépéseket és metsző síkokat alkalmaz a manuálisan elkészített testeken annak érdekében, hogy nagyszámú Mental Cutting Test szcenáriót állíthassunk elő.*

A második fejezet a szkript-alapú módszer kimenetének utófeldolgozásával és elemzésével foglalkozott. Elsőként egy olyan eljárást javasoltunk, mely alkalmas a Blender által kiszámolt metszetek hibáinak detektálására és javítására. Később egy *oszd meg és uralkodj!* algoritmust terveztünk az MCT feladatok megoldásaiként előállított alakzatok egyedi halmazának meghatározására. Ez a fejezet egy tézist tartalmazott a 2022-ben megjelent konferenciacikk (R. Tóth, B. Tóth, Zichar és tsai., 2023a) alapján:

3. *Elvégeztük a szkript-alapú eljárás kimeneteként előállított SVG assetek utófeldolgozását és rendellenességeket észleltünk. A manuálisan elvégzett ellenőrzés tapasztalatai alapján megterveztük és implementáltuk a validálás automatizált változatát. Az algoritmus képes a számítási pontatlanságok következtében eltérésekkel rendelkező assetek automatikusan javításra vagy eldobására.*

Következőként az adathalmazzal kapcsolatos egyik legfontosabb kérdésre fókuszáltunk: *Hogyan lehet a szcenáriókat hatékonyan tárolni és valós időben kiszolgálni különböző felhasználási szándékkal?*. A kérdésre egy olyan többszintű kódolási séma kidolgozásával adtunk választ, mely az assetek veszteségmentes tárolását 10 000-szeres tömörítési arány és 99,99%-os helymegtakarítás mellett teszi lehetővé. Mindezek mellett az assetek generálását gyorsabbá is tettük a Blenderben végzett számítások minimalizálásával. A fejezet egy tézist tartalmazott a 2023-ban megjelent, Q2-es besorolású és 3,0 impakt faktorú folyóiratcikk (R. Tóth, Miklós Hoffmann és Zichar, 2023a) alapján:

4. *Bevezettünk egy olyan kódolási sémát, mely négy szintből áll és az adathalmaz tárolását támogatja, jelentősen csökkentve annak méretét. A kidolgozott módszer az assetek egy részhalmazát kódolja, amiből képes a teljes adathalmazt visszafejteni a Blender számításaihoz képest 3% idő komplexitással, meghaladva a 10 000-szeres tömörítési arányt és a 99,99%-os helymegtakarítást.*

Utolsóként a fő tudományos eredményeink lehetséges felhasználására adtunk példákat; a különféle alkalmazások fejlesztése során gyűjtött tapsztalatok ugyanis fontos szerepet játszottak a permutálás és a kódolás megtervezésében, továbbá tesztelni és bizonyítani is az alkalmazások használatával tudtuk az eljárások alkalmazhatóságát. Ez a fejezet egy tézist tartalmazott négy konferenciacikk és két konferenciaelőadás (R. Tóth, Miklós Hoffmann, Kósa és Zichar, 2019; R. Tóth, Zichar és Miklós Hoffmann, 2020; R. Tóth, Zichar és Miklós Hoffmann, 2021; R. Tóth, B. Tóth, Zichar és tsai., 2023b; R. Tóth, Miklós Hoffmann, Kósa és Zichar, 2020; R. Tóth, Miklós Hoffmann és Zichar, 2023b) alapján:

5. *Alkalmazásokat fejlesztettünk, melyek a saját eljárásainkkal előállított, utófeldolgozott és kódolt szcenáriókat használják. Az alkalmazások bizonyítják azt, hogy az adathalmazunk szcenáriói alkalmasak MCT feladatok különböző céllal történő, valósidejű összeállítására és kiszolgálására, 2D-ben és 3D-ben egyaránt.*

Köszönetnyilvánítás

Elsőként témavezetőmnek, Zichar Mariannak szeretném megköszönni azt a rengeteg segítséget, melyet a PhD kutatásom és valamennyi más, ügyes-bajos egyetemi munkám kapcsán kaptam tőle. Ugyancsak szeretném megköszönni Hoffmann Miklósnak az eredeti kutatási ötletet, valamint azt, hogy iránymutatásaival segített.

Köszönöm a családom kitartó támogatását, különösen testvéremnek, Tóth Bálintnak a társszerzőséget is érő segítségét a metszetek validálása és a cikkek nyelvi lektorálása során. Ugyancsak szeretném kiemelni Bernáth Mónikát, Nagy Dórát, Nagy Mártát, Orosz Anettet, Varga Viktort és Vécsi Ádámot, akik számtalamszor véleményezték a munkámat, esetleg ötleteikkel segítettek az előttem álló feladatok megvalósításában.

Köszönet illeti továbbá az Informatikai Kar azon jelenlegi és volt oktatóit, akik egykor alap- és mesterszakos hallgatóként alapozták meg a szakmai tudásomat, különféle tudományterületeket képviselve. Örömmel tölt el, hogy a szakmai tudásomhoz leginkább hozzájárult oktatókkal előbb-utóbb kollegaként is együtt dolgozhattam előbb demonstrátorként, majd pedig külsős óraadóként, esetleg megpróbálhattam a tárgyaik átvétele után is fenntartani azok szakmai színvonalát. Ugyancsak szeretnék köszönetet mondani a Dékáni Hivatal azon dolgozóinak is, akik segítettek engem a pályázatokkal és az oktatással kapcsolatos ügyek intézésében.

References

[1] Dayana Farzeeha Ali et al. „Overcoming the problems faced by student's in learning engineering drawing with the implementation of augmented reality learning environment". In: *Man in India* 97.17 (2017), pp. 147–159.

[2] F. Alpiste Penalba, J. Torner Ribé, and M.A. Brigos Hermida. „Exploring Virtual Reality to improve engineering students' spatial abilities. Pilot study". In: *EDULEARN19 Proceedings*. 11th International Conference on Education and New Learning Technologies. Palma, Spain: IATED, July 2019, pp. 6275–6284. ISBN: 978-84-09-12031-4. DOI: 10.21125/edulearn.2019.1503.

[3] Bruno Alves et al. „On Capitalizing on Augmented Reality to Impart Solid Geometry Concepts: An Experimental Study". In: *Universal Access in Human–Computer Interaction. Designing Novel Interactions*. Ed. by Margherita Antona and Constantine Stephanidis. Cham: Springer International Publishing, 2017, pp. 105–117. ISBN: 978-3-319-58703-5. DOI: 10.1007/978-3-319-58703-5_8.

[4] Holly K Ault and Samuel John. „Assessing and enhancing visualization skills of engineering students in Africa: A comparative study". In: *The Engineering Design Graphics Journal* 74.2 (2010).

[5] Guillaume Ayoub. *CairoSVG*. en. https://pypi.org/project/CairoSVG/. Accessed: 2023-08-13.

[6] Gabriel Barata, Sandra Gama, Joaquim Jorge, and Daniel Gonçalves. „Improving participation and learning with gamification". In: *Proceedings of the First International Conference on Gameful Design, Research, and Applications*. New York, NY, USA: ACM, 2013. DOI: 10.1145/2583008.2583010.

[7] David Ben-Chaim, Glenda Lappan, and Richard T. Houang. „The Effect of Instruction on Spatial Visualization Skills of Middle School Boys and Girls". In: *American Educational Research Journal* 25.1 (Jan. 1988), pp. 51–71. DOI: 10.3102/00028312025001051.

[8] Alan J. Bishop. „Spatial abilities and mathematics education – A review". In: *Educational Studies in Mathematics* 11.3 (Aug. 1980), pp. 257–269. ISSN: 1573-0816. DOI: 10.1007/BF00697739.

[9] Katerina Bogomolova et al. „The Effect of Stereoscopic Augmented Reality Visualization on Learning Anatomy and the Modifying Effect of Visual-Spatial Abilities: A Double-Center Randomized Controlled Trial". In: *Anatomical Sciences Education* 13.5 (Jan. 2020), pp. 558–567. DOI: 10.1002/ase.1941.

[10] Nina Bohlmann and Ralf Benölken. „Complex Tasks: Potentials and Pitfalls". In: *Mathematics* 8.10 (2020). ISSN: 2227-7390. DOI: 10.3390/math8101780.

[11] Ágnes Bosnyák and Rita Nagy-Kondor. „The Spatial Ability and Spatial Geometrical Knowledge of University Students Majored in Mathematics". In: (Jan. 2008), pp. 1–25.

[12] Attila Bölcskei, Szilvia Gál-Kállay, Andras Z Kovács, and Csilla Sörös. „Development of Spatial Abilities of Architectural and Civil Engineering Students in the Light of the Mental Cutting Test". In: *Journal for Geometry and Graphics* 16.1 (2012), pp. 103–115.

[13] Kevin Browne and Christopher Anand. „Gamification and serious game approaches for introductory computer science tablet software". In: *Proceedings of the First International Conference on Gameful Design, Research, and Applications*. New York, NY, USA: ACM, 2013. DOI: 10.1145/2583008.2583015.

[14] Carlos Carbonell-Carrera and Jose Luis Saorin. „Virtual Learning Environments to Enhance Spatial Orientation". In: *EURASIA Journal of Mathematics, Science and Technology Education* 14.3 (Nov. 2017). DOI: 10.12973/ejmste/79171.

[15] CEEB. *CEEB Special Aptitude Test in Spatial Relations (MCT). Developed by the College Entrance Examination Board*. 1939.

[16] Francisco del Cerro Velázquez and Ginés Morales Méndez. „Application in Augmented Reality for Learning Mathematical Functions: A Study for the Development of Spatial Intelligence in Secondary Education Students". In: *Mathematics* 9.4 (2021), p. 369. DOI: 10.3390/math9040369.

[17] Eunhee Chang, Hyun Taek Kim, and Byounghyun Yoo. „Virtual reality sickness: A review of causes and measurements". en. In: *Int. J. Hum. Comput. Interact.* 36.17 (2020), pp. 1658–1682. DOI: 10.1080/10447318.2020.1778351.

[18] Yu-ching Chen. „Effect of Mobile Augmented Reality on Learning Performance, Motivation, and Math Anxiety in a Math Course". In: *Journal of Educational Computing Research* 57.7 (2019), pp. 1695–1722. DOI: 10.1177/0735633119854036.

[19] Cheryl A. Cohen and Mary Hegarty. „Inferring cross sections of 3D objects: A new spatial thinking test". In: *Learning and Individual Differences* 22.6 (2012), pp. 868–874. ISSN: 1041-6080. DOI: 10.1016/j.lindif.2012.05.007.

[20] Merryn Cole et al. „The Relationship between Spatial Ability and the Conservation of Matter in Middle School". In: *Education Sciences* 11.1 (2021). ISSN: 2227-7102. DOI: 10.3390/educsci11010004.

[21] Adrienne Decker and Elizabeth Lane Lawley. „Life's a game and the game of life: How making a game out of it can change student behavior". In: *Proceeding of the 44th ACM technical symposium on Computer science education - SIGCSE '13*. New York, New York, USA: ACM Press, 2013. DOI: 10.1145/2445196.2445269.

[22] Xuan Di and Xudong Zheng. „A meta-analysis of the impact of virtual technologies on students' spatial ability". In: *Educational technology research and development* 70.1 (Jan. 2022), pp. 73–98. DOI: 10.1007/s11423-022-10082-3.

[23] Anne Estapa and Larysa Nadolny. „The effect of an augmented reality enhanced mathematics lesson on student achievement and motivation". In: *Journal of STEM education* 16.3 (2015).

[24] Míriam Flores-Bascuñana, Pascual D Diago, Rafael Villena-Taranilla, and Dionisio F Yáñez. „On Augmented Reality for the learning of 3D-geometric contents: A preliminary exploratory study with 6-Grade primary students". In: *Education Sciences* 10.1 (2020), p. 4. DOI: 10.3390/educsci10010004.

[25] Juan Garzón, Juan Pavón, and Silvia Baldiris. „Systematic review and meta-analysis of augmented reality in educational settings". In: *Virtual Reality* 23.4 (Feb. 2019), pp. 447–459. DOI: 10.1007/s10055-019-00379-9.

[26] Zeynep Gecu-Parmaksiz and Ömer Delialioğlu. „The effect of augmented reality activities on improving preschool children's spatial skills". In: *Interactive Learning Environments* 28.7 (Nov. 2018), pp. 876–889. DOI: 10.1080/10494820.2018.1546747.

[27] Andri Gerber, ed. *Spatial Abilities. A Workbook for Students of Architecture*. Basel: Birkhäuser Verlag GmbH, 2020. ISBN: 978-3-0356-2043-6.

[28] R Gorska, Sheryl Sorby, and C Leopold. „Gender differences in visualization skills - An international perspective". In: *The Engineering Design Graphics Journal* 62.3 (1998), p. 10.

[29] Tibor Guzsvinecz, Csaba Kovacs, et al. „Developing a virtual reality application for the improvement of depth perception". In: *2018 9th IEEE International Conference on Cognitive Infocommunications (CogInfoCom)*. 2018, pp. 000017–000022. DOI: 10.1109/CogInfoCom.2018.8639935.

[30] Tibor Guzsvinecz, Éva Orbán-Mihálykó, Cecília Sik-Lányi, and Erika Perge. „Investigation of spatial ability test completion times in virtual reality using a desktop display and the Gear VR". In: *Virtual Reality* 26.2 (Mar. 2021), pp. 601–614. DOI: 10.1007/s10055-021-00509-2.

[31] Tibor Guzsvinecz, Cecilia Sik-Lanyi, Eva Orban-Mihalyko, and Erika Perge. „Implementation of the Heinrich Spatial Visualization Test in a Virtual Environment". In: *International Journal of Engineering and Management Sciences* 6.4 (Dec. 2021). DOI: 10.21791/ijems.2021.4.1..

[32] Ezgi Tosik Gün and Bilal Atasoy. „The Effects of Augmented Reality on Elementary School Students' Spatial Ability and Academic Achievement". In: *TED EĞİTİM VE BİLİM* (Aug. 2017). DOI: 10.15390/eb.2017.7140.

[33] Nathan W. Hartman et al. „Virtual reality-based spatial skills assessment and its role in computer graphics education". In: *ACM SIGGRAPH'06 Proceedings*. 2006, p. 46. DOI: 10.1145/1179295.1179342.

[34] Omar Huerta et al. „An approach to improve technical drawing using VR and AR tools". In: *Computer-Aided Design and Applications* 17.4 (Nov. 2019), pp. 836–849. DOI: 10.14733/cadaps.2020.836-849.

[35] Duy Huynh, Long Zuo, and Hiroyuki Iida. „An assessment of game elements in language-learning platform duolingo". In: *2018 4th International Conference on Computer and Information Sciences (ICCOINS)*. IEEE, 2018. DOI: 10.1109/ICCOINS.2018.8510568.

[36] Karl M Kapp. *The gamification of learning and instruction: Game-based methods and strategies for training and education*. en. Nashville, TN: John Wiley & Sons, 2012.

[37] R. Llorens, M. Contero, and M. Alcañiz. „Evolution of Spatial Ability in Freshman Engineering Students: A comparison between 2012 and 2019 Cohorts". In: *EDULEARN20 Proceedings*. 12th International Conference on Education and New Learning Technologies. Online Conference: IATED, July 2020, pp. 7172–7179. ISBN: 978-84-09-17979-4. DOI: 10.21125/edulearn.2020.1844.

[38] R. Llorens, J. Martín Gutiérrez, M. Contero, and M. Alcañiz. „Design, Usage and Implementation of an Augmented Reality-based App Designed to Train Spatial Skills". In: *EDULEARN20 Proceedings*. 12th International Conference on Education and New Learning Technologies. Online Conference: IATED, July 2020, pp. 7074–7080. ISBN: 978-84-09-17979-4. DOI: 10.21125/edulearn.2020.1824.

[39] R. Llorens Rodríguez et al. „Design of a Competitive Multi-Player Mobile Game to Train Spatial Skills". In: *INTED2022 Proceedings*. 16th International Technology, Education and Development Conference. Online Conference: IATED, Mar. 2022, pp. 4106–4112. ISBN: 978-84-09-37758-9. DOI: 10.21125/inted.2022.1119.

[40] Ian Lochhead, Nick Hedley, Arzu Çöltekin, and Brian Fisher. „The Immersive Mental Rotations Test: Evaluating Spatial Ability in Virtual Reality". In: *Frontiers in Virtual Reality* (2022), p. 5. DOI: 10.3389/frvir.2022.820237.

[41] Bruce R Maxim, Stein Brunvand, and Adrienne Decker. „Use of role-play and gamification in a software project course". In: *2017 IEEE Frontiers in Education Conference (FIE)*. IEEE, 2017. DOI: 10.1109/FIE.2017.8190501.

[42] MozDevNet. *SVG: Scalable Vector Graphics*. en. Accessed: 2023-08-13. URL: https://developer.mozilla.org/en-US/docs/Web/SVG.

[43] Rita Nagy-Kondor and Saeed Esmailnia. „Polyhedrons vs. curved surfaces with mental cutting: impact of spatial ability". In: *Acta Polytechnica Hungarica* 18.6 (2021), pp. 71–83. DOI: 10.12700/APH.18.6.2021.6.4.

[44] B Nemeth and M Hoffmann. „Gender differences in spatial visualization among engineering students". In: *Annales Mathematicae et Informaticae* 33 (2006), pp. 169–174. ISSN: 1787-5021.

[45] Brigitta Németh, Csilla Sörös, and Miklós Hoffmann. „Typical mistakes in Mental Cutting Test and their consequences in gender differences". In: *Teaching Mathematics and Computer Science* 5.2 (2007), pp. 385–392. DOI: 10.5485/TMCS.2007.0169.

[46] Christos Papakostas, Christos Troussas, Akrivi Krouska, and Cleo Sgouropoulou. „Exploration of Augmented Reality in spatial abilities training: A systematic literature review for the last decade". en. In: *Informatics in Education* 20.1 (Mar. 2021), pp. 107–130. DOI: 10.15388/infedu.2021.06.

[47] T Parsons. „Sex differences in mental rotation and spatial rotation in a virtual environment". In: *Neuropsychologia* 42.4 (2004), pp. 555–562. DOI: 10.1016/j.neuropsychologia.2003.08.014.

[48] Plamen D Petrov and Tatiana V Atanasova. „The Effect of Augmented Reality on Students' Learning Performance in Stem Education". In: *Information* 11.4 (2020), p. 209. DOI: 10.3390/info11040209.

[49] E. Ponick and A. Stuckenholz. „Impact of user types in gamified learning environments". In: *INTED2019 Proceedings*. 13th International Technology, Education and Development Conference. Valencia, Spain: IATED, Mar. 2019, pp. 2419–2425. ISBN: 978-84-09-08619-1. DOI: 10.21125/inted.2019.0669.

[50] Norma Presmeg. „Spatial Abilities Research as a Foundation for Visualization in Teaching and Learning Mathematics". In: *Critical Issues in Mathematics Education: Major Contributions of Alan Bishop*. Ed. by Philip Clarkson and Norma Presmeg. Boston, MA: Springer US, 2008, pp. 83–95. ISBN: 978-0-387-09673-5. DOI: 10.1007/978-0-387-09673-5_6.

[51] Norma Presmeg. „Visualization and Learning in Mathematics Education". In: *Encyclopedia of Mathematics Education*. Ed. by Stephen Lerman. Cham: Springer International Publishing, 2020, pp. 900–904. ISBN: 978-3-030-15789-0. DOI: 10.1007/978-3-030-15789-0_161.

[52] James Purnama, Daniel Andrew, and Maulahikmah Galinium. „Geometry learning tool for elementary school using augmented reality". In: *2014 International Conference on Industrial Automation, Information and Communications Technology* (2014), pp. 145–148. DOI: 10.1109/IAICT.2014.6922112.

[53] Python Software Foundation. *struct — Interpret bytes as packed binary data — Python 3.11.4 documentation*. https://docs.python.org/3/library/struct.html. Accessed: 2023-08-13. 2022.

[54] Python Software Foundation. *xml.dom.minidom — Minimal DOM implementation — Python 3.11.4 documentation*. https://docs.python.org/3/library/xml.dom.minidom.html. Accessed: 2023-08-13.

[55] Claudia Quaiser-Pohl. „The Mental Cutting Test "Schnitte" and the Picture Rotation Test-Two New Measures to Assess Spatial Ability". In: *International Journal of Testing* 3.3 (Sept. 2003), pp. 219–231. DOI: 10.1207/s15327574ijt0303_2.

[56] G. Raju, S. Sorby, and C. Reid. „INVESTIGATING THE RELATIONSHIP BETWEEN SPATIAL SKILLS AND ENGINEERING DESIGN". In: *ICERI2022 Proceedings*. 15th annual International Conference of Education, Research and Innovation. Seville, Spain, Nov. 2022, pp. 421–426. ISBN: 978-84-09-45476-1. DOI: 10.21125/iceri.2022.0144.

[57] Albert A Rizzo et al. „The virtual reality mental rotation spatial skills project". In: *CyberPsychology & Behavior* 1.2 (1998), pp. 113–119.

[58] Parviz Safadel and David White. „Effectiveness of computer-generated virtual reality (VR) in learning and teaching environments with spatial frameworks". In: *Applied Sciences* 10.16 (2020), p. 5438. DOI: 10.3390/app10165438.

[59] Dimitrios Saredakis et al. „Factors associated with virtual reality sickness in head-mounted displays: A systematic review and meta-analysis". en. In: *Front. Hum. Neurosci.* 14 (2020), p. 96. DOI: 10.3389/fnhum.2020.00096.

[60] Zohreh Shaghaghian, Heather Burte, Dezhen Song, and Wei Yan. „Learning Geometric Transformations for Parametric Design: An Augmented Reality (AR)-Powered Approach". In: *Computer-Aided Architectural Design. Design Imperatives: The Future is Now*. Ed. by David Gerber et al. Singapore: Springer Singapore, 2022, pp. 515–527. ISBN: 978-981-19-1280-1. DOI: 10.1007/978-981-19-1280-1_31.

[61] Željka Milin Šipuš and A Cižmešija. „Spatial ability of students of mathematics education in Croatia evaluated by the Mental Cutting Test". In: *Annales Mathematicae et Informaticae* 40 (2012), pp. 203–216.

[62] Sheryl A. Sorby. „Eucational Research in Developing 3-D Spatial Skills for Engineering Students". In: *International Journal of Science Education* 31.3 (2009), pp. 459–480. DOI: 10.1080/09500690802595839.

[63] Thomas Suselo, Burkhard C Wünsche, and Andrew Luxton-Reilly. „Using Mobile Augmented Reality for Teaching 3D Transformations". In: *Proceedings of the 52nd ACM Technical Symposium on Computer Science Education*. 2021, pp. 872–878. DOI: 10.1145/3408877.3432401.

[64] Jakub Swacha and Karolina Muszyńska. „Design patterns for gamification of work". In: *Proceedings of the Fourth International Conference on Technological Ecosystems for Enhancing Multiculturality - TEEM '16*. New York, New York, USA: ACM Press, 2016. DOI: 10.1145/3012430.3012604.

[65] Arthur Tang, Charles Owen, Frank Biocca, and Weimin Mou. „Comparative effectiveness of augmented reality in object assembly". In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM, Apr. 2003. DOI: 10.1145/642611.642626.

[66] The Khronos® 3D Formats Working Group. *GlTF™ 2.0 specification - Version 2.0.1*. en. https://registry.khronos.org/glTF/specs/2.0/glTF-2.0.html. Accessed: 2023-08-13. 2021.

[67] Maria Grazia Tosto et al. „Why do spatial abilities predict mathematical performance?" In: *Developmental Science* 17.3 (2014), pp. 462–470. DOI: 10.1111/desc.12138.

[68] Christos Troussas, Akrivi Krouska, and Maria Virvou. „Reinforcement theory combined with a badge system to foster student's performance in e-learning environments". In: *2017 8th International Conference on Information, Intelligence, Systems & Applications (IISA)*. IEEE, 2017. DOI: 10.1109/IISA.2017.8316421.

[69] W. Zimmermann, S. Cunningham, and Mathematical Association of America. Committee on Computers in Mathematics Education. *Visualization in Teaching and Learning Mathematics: A Project*. MAA Notes and reports series. Mathematical Association of America, 1991. ISBN: 9780883850718.

A. List of relevant publications

Referred journal papers

[J1] Róbert Tóth. „Script-aided generation of Mental Cutting Test exercises using Blender". In: *Annales Mathematicae et Informaticae* 54 (2021), pp. 147–161. DOI: 10.33039/ami.2021.03.011.

[J2] Róbert Tóth, Miklós Hoffmann, and Marianna Zichar. „Lossless Encoding of Mental Cutting Test Scenarios for Efficient Development of Spatial Skills". In: *Education Sciences* 13.2 (2023). ISSN: 2227-7102. DOI: 10.3390/educsci13020101.

[J3] Róbert Tóth, Bálint Tóth, Miklós Hoffmann, and Marianna Zichar. „`viskillz-blender` - A Python package to generate assets of Mental Cutting Test exercises using Blender". In: *SoftwareX* 22 (2023), p. 101328. ISSN: 2352-7110. DOI: 10.1016/j.softx.2023.101328.

Papers in conference proceedings

[I1] Róbert Tóth, Miklós Hoffmann, Márk Kósa, and Marianna Zichar. „Játékosított alkalmazások a programozási ismeretek és a térlátási képességek fejlesztésére". In: *A 16 éves PEME XIX. PhD-Konferenciájának előadásai*. Budapest, 2019, pp. 358–365.

[I2] Róbert Tóth, Bálint Tóth, Marianna Zichar, et al. „Detecting and Correcting Errors in Mental Cutting Test Intersections Computed with Blender". In: *ICGG 2022 - Proceedings of the 20th International Conference on Geometry and Graphics*. Cham: Springer International Publishing, 2023, pp. 904–916. DOI: doi.org/10.1007/978-3-031-13588-0_79.

[I3] Róbert Tóth, Bálint Tóth, Marianna Zichar, et al. „Educational Applications to Support the Teaching and Learning of Mental Cutting Test Exercises". In: *ICGG 2022 - Proceedings of the 20th International Conference on Geometry and Graphics*. Cham: Springer International Publishing, 2023, pp. 928–938. ISBN: 978-3-031-13588-0. DOI: doi.org/10.1007/978-3-031-13588-0_81.

[I4] Róbert Tóth, Marianna Zichar, and Miklós Hoffmann. „Gamified Mental Cutting Test for enhancing spatial skills". In: *2020 11th IEEE International Conference on Cognitive Infocommunications (CogInfoCom)*. 2020, pp. 000299–000304. DOI: 10.1109/CogInfoCom50765.2020.9237888.

[I5] Róbert Tóth, Marianna Zichar, and Miklós Hoffmann. „Improving and Measuring Spatial Skills with Augmented Reality and Gamification". In: *ICGG 2020 - Proceedings of the 19th International Conference on Geometry and Graphics*. Cham: Springer International Publishing, 2021, pp. 755–764. DOI: https://doi.org/10.1007/978-3-030-63403-2_68.

Book chapters

[B1] Róbert Tóth and Marianna Zichar. „Augmented and Gamified Lives". In: *Encyclopedia of Computer Graphics and Games*. Ed. by Newton Lee. Cham: Springer International Publishing, 2020, pp. 1–6. ISBN: 978-3-319-08234-9. DOI: 10.1007/978-3-319-08234-9_89-1.

Conference talks

[C1] Róbert Tóth, Miklós Hoffmann, Márk Kósa, and Marianna Zichar. „Motivating students' study with ICT". ICAI 2020 11th International Conference on Applied Informatics. Eger, 2020. URL: https://icai.uni-eszterhazy.hu/2020/abstracts/ICAI_2020_abstract_093.pdf.

[C2] Róbert Tóth, Miklós Hoffmann, and Marianna Zichar. „viSkillz – a project dealing with Mental Cutting Test exercises". ICAI 2023 12th International Conference on Applied Informatics. Eger, 2023. URL: https://icai.uni-eszterhazy.hu/2023/abstracts/ICAI_2023_abstract_065.pdf.