Conference on Information Technology and Data Science

A WebGL-based Virtual Puzzle Game for Spatial Skill Development Purposes*

Bence Dániel Erős[ab], **Roland Kunkli**[b]

[a]University of Debrecen, Doctoral School of Informatics

[b]University of Debrecen, Faculty of Informatics

{eros.bence,kunkli.roland}@inf.unideb.hu

Proceedings of the 1st Conference on Information Technology and Data Science
Debrecen, Hungary, November 6–8, 2020
published at http://ceur-ws.org

Abstract

Because of their unique challenges, twisty puzzles and similar logical games are popular with children and adults alike. Therefore, they are one of the key tools of skill development in public opinion since their first appearance. The physical environment creates specific limitations to these games, but new tools from computer graphics and its virtual environments allow us to reach new possibilities.

We introduce an application that helps the users to try rotations on 3D models based on the rules of the well-known puzzle Rubik's Cube. In our solution, these models could be different from the usually used symmetric and convex shapes.

Keywords: Virtual spatial puzzle, Rubik's Cube, skill development, WebGL

AMS Subject Classification: 00A09, 97G30, 97R60

1. Introduction

The first phases of skill development start in the early stages of our life when we learn how to interact with our environment efficiently. All these stages can

*This work was supported by the construction EFOP-3.6.3-VEKOP-16-2017-00002. The project was supported by the European Union, co-financed by the European Social Fund.

prepare us for adulthood, where we can easily solve all the tasks of our everyday lives. For decades, researchers have been interested in how they could develop and measure specific skills. One related example is the relationship between educational psychology and spatial skills. Today a skill development direction has emerged in science, where the combination of classical tests and new technological opportunities provides new methods for us. For example, the work by Tóth et al. [13], where the authors introduce the classical Mental Cutting Test enriched with features from AR. The work implements principles with elements delivered from games also, according to the so-called gamification.

However, games and playing can be a primary method for this skill developing process, on their own. With playing, we can discover our environment and make explorations about our reality, and here learning happens in a fun way, as a pleasurable and entertaining process [5].

From the related research by Caillois [3], we can see that not every kind of game is suitable for skill development purposes. In his work, the author creates a categorization for games to examine the nature of these activities. The categories are based on two attributes, the mode and the motive of playing. On the one hand, the mode has determined by the quantities of rules. Here we can meet with the concept of *paidia*, which means the arbitrary, uncontrolled form of playing. On the opposite side, there is *ludus*, where playing appears in a deliberated, rule-based form. When we analyze the motive of playing with Caillois's systematization, we can find four different—but more or less overlapping—segments. The first two are *agôn*, the games of skills and competition (chess or football), and *alea*, where games only claim the faith of luck (lottery). A third one is *mimicry*, which includes all theatrical interpretations and behavioral changes. The fourth category in his system is *ilinx*, where all activities help us to experience the fun and save form of panic, fall, or fast speed (skiing, roller coaster).

With this knowledge, we can discover a wide range of games to start a plan for our application. At this point, the question can be, what the benefits are of the virtual implementation of our selected game, compared to its physical form [9]. Considering the limitations of both games and computers or other devices, we can find properties that make some adaptation less enjoyable in a new, virtual form than in the original case. A computer probably has limited memory or response time and always has a 2D display. Even so, the virtual environment can help us improve the original interface. It can teach the rules of the game, or it can give a hint to help the player who tries to reach the goal whenever some difficulties are appearing [9].

In this work, we have selected spatial puzzle games. The goal of these games is easily understandable and acceptable to everyone, like in the case of the well-known spatial puzzle toy, the *Rubik's Cube*. The following paragraphs analyze our choice through the aspects mentioned earlier:

- **A game of abilities.** The success or failure of finding a solution depends only on the player's abilities and efforts (*agôn*) [3]. Even though if the players trust the luck of some random rotations, if they are not persistent enough, or they do

not spend enough time memorizing algorithmic steps, they can easily get lost in the variety of shuffles.

- **Developing new skills.** For today, research has shown that spatial puzzle games have a measurable skill-developing effect on our visual-spatial skills [5]. These games can affect those areas in our brains, which help us to create a mental representation of the objects in our environment. When we develop this ability, we can also express our ideas and thoughts in drawing with the employment of spatial concepts [5, 16]. In this regard, these puzzle games can help us to achieve our skill-developing purposes in our application.

- **The physical limitations are eliminated.** Twisty puzzles contain a well-designed inner mechanism inside of their central part, which is responsible for two main tasks: this part interlocks all of the puzzle pieces, and at the same time, it allows us to perform rotations on our puzzle. In terms of the shape of the puzzle, we can face new challenges and limitations also [12]. These requirements will be mentioned in Section 2.2 in more detail. In our work, we have tried to relieve the burden of these challenges, and we have done experiments to try new possibilities beyond physical boundaries, such as floating puzzle pieces.

- **3D technologies in education result in good practice.** 3D content has gained a victory on consumer culture with 3D movies, software, and other 3D products. With this in mind, no wonder that research has started to observe the effectiveness of these 3D technologies on societies and education. Today, it is clear that the benefits of these technologies exist, and the necessity of them in education is not a yes or no question. However, the employment of some technologies, unfortunately, looks unavailable for public education these days. In their work, Papp et al. [11] have collected the main benefits and limitations of 3D printers in education. Here, the authors characterized 3D printers as expensive technologies. In this regard, a 3D web application looks like a much more realizable product. Moreover, people have become confident users of web applications in the digital era.

- **Help in solution.** A unique advantage of the computer is that it is available at any time as a patient teacher. If the player has uncertainty about the rules of the game, or if the solution is hard to find, an algorithm with an interactive interface can help us learn to overcome the difficulties [9]. In our application, the user can generate the steps of a complete solution path.

- **Giving up the tactile experience.** It is quite evident that a 3D toy cannot be translated into our 2D screen perfectly. In the case of spatial puzzle games, we will face the loss of the simplicity of rotations by hand, and we cannot naturally reach any point of view [9]. However, technology can reduce this drawback because modern 3D graphical libraries provide more and more tools to create the best possible spatial virtual environments. These libraries will be mentioned in Section 2.1 in more detail.

These thoughts (with, keep in mind, the limitations from the last item) make it a reasonable decision to build a web application based on the idea of spatial puzzles. In the next sections, we will introduce more details about the technical and theoretical background of our work. In Section 2, we introduce some other research cases about specific graphical problems. In these experiments, similarly for this work, the WebGL-based Three.js library [2] or the Blender graphical toolset [1] have been used. This section also introduces some motive for our virtual puzzle design plans. Later, in Section 3, we detail the concept of permutation group and its connection to our implemented solution algorithm, which appears in the application to help the user. The last main part, Section 4, is a guide from our prototype to the overview of the full application. After the conclusion in Section 5, this paper ends with Section 6, which is a quick insight into the possible directions of future software versions.

2. Related Work

The first subsection of this part collects some examples, where technologies like WebGL and the Blender graphical toolset were useful for researchers to solve their problems. Later in this section, we describe the closely related research topic of puzzle design. Although these works search for solutions to challenges which mainly resulted from the physical environment and its laws, we can find one of the best bases of comparison here for a virtual puzzle application.

2.1. 3D Web Applications with WebGL and Blender

In the last decade, the creation and usage of 3D web applications have significantly changed. Before the WebGL API, the multimedia experience on the web had got limitations by its plug-in-based nature. In those years, users had to download an external applet just to reach the 3D graphical content. This method needed a burden of download time and extra storage space that resulted in a struggle by users and mobile devices with limited capacity also [4, 7].

Later, 3D content could move into web browsers. Based on OpenGL ES 2.0 by Khronos Group, the WebGL platform is in conjunction with web development technologies. The drawn context appears in an HTML5 *<canvas>* element as part of a standard HTML page. Therefore technologies, like CSS (Cascading Style Sheet), or JavaScript, can provide additional options. In this way, the content is not just a replicate of a desktop application, but it can extract the real power of the web experience [4, 7].

Even though more and more browsers supported WebGL, and a broader range of users could enjoy its results, in the early years, content production was still a challenge. The low-level API required advanced knowledge of linear algebra and other related mathematics concepts, as well as shading. This complex flow starts from the define and load of vertex and fragment data and lasts using model-view-projection matrices. The democratization of this process has come in with the

appearance of higher-level libraries [4]. For today, many libraries provide easier access to the tools of WebGL, for example, GLGE, SceneJS, or Three.js. In addition, to supporting the general 3D content production, some libraries have developed for problem-specific purposes; this is the case of Babylon.js for game development or 3Dmol.js for scientific data visualization.

This diversity of existing libraries can reflect the most different fields of 3D web applications. The main groups besides video games and data visualization are engineering, architecture, heritage, as well as information systems, e-learning, and many more [4].

These libraries have been presented in research from the early stages. In their paper from the beginning of the last decade, Hering et al. [7] have described WebGL as new and promising technology. Their work introduces a virtual campus as an information system about persons, rooms, and facilities. The work uses GLGE for drawing, but the process of building model creation is provided by the tools of open source project, Blender. This mentioned research reports an exporter to handle graphical data from Blender to the web application.

The relationship of Blender and a WebGL-based virtual campus appears in the work of Yuniarti et al. [15] also. The work uses the Three.js library, a more popular JavaScript library, compared to GLGE.

An example of a web application for educational purposes is presented in [10]. The work details the creation steps of a 3D hydraulic plant from the topics of control theory with the various tools of Blender and the Three.js library.

2.2. The Physical Design of Puzzles

With the knowledge of tools for the virtual puzzle preparation, at this point, there is still a question mark over the mechanics. Before we experiment with the possibilities of virtual mechanics, we look through the experiences from the physical design. The requirements of puzzle design appear in the research of Sun and Zheng [12] in the following details:

- **The requirement of interlocked pieces:** The design of a new puzzle still requires expert knowledge because of the need for rotatable pieces, and with an inaccurate plan, the construction may fall apart.
- **The requirement of rotation about axes:** The main problem here is blocking. Without the symmetry or convexity properties, every shape can stop the playing process when some pieces of the puzzle block each other due to collision. A presented method in [12] handles this problem with some deformation steps of the original model before the physical printing process to avoid the collision of puzzle pieces.

3. Mathematical Background of the Rubik's Cube

This section introduces the connection between the mathematical group theory and the Rubik's Cube, which helps us describe any manipulation of a spatial puzzle.

3.1. The $3 \times 3 \times 3$ Rubik's Cube

While we are holding a $3 \times 3 \times 3$ Rubik's Cube in our hand, we can see that it consists of twenty-six little pieces that have interlocked. There are eight corner pieces with three colored facets, twelve edge pieces with two colored facets, and six face pieces that have only one colored facet. To shuffle the original order of the colored facets, we can rotate any face of the cube. Rotations in a clockwise direction or counterclockwise direction are both performable. All of these twelve rotations will generate a valid state of the cube.

The reason behind this validation comes from group theory, where we can define a set of G, that consists of every possible permutation of the puzzle as elements. The group operator $*$ can be the rotation of a face. The correspondence between this $(G; *)$ object and the axioms of the mathematical group is in [6].

Considering these properties, we can store the current shuffle state of the puzzle in a compound data collection, for example, in a list or a string. Once we have stored the information of the shuffle which the player has generated, we can create a sequence of steps to find the solution. In this sequence, every rotation can be summarized in general as a permutation $f\colon \mathbb{Z}_n \to \mathbb{Z}_n$ by a $2 \times n$ array [8]. During the game, these rearrangements can easily navigate us towards the solution.

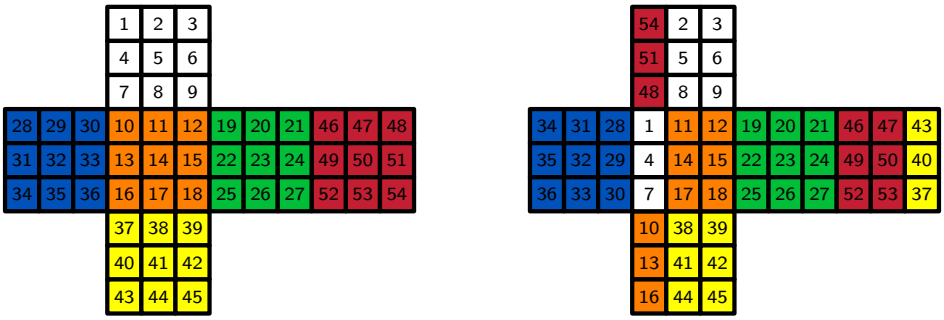

Figure 1. The left side of the figure shows a possible index order of the facets in the case of the $3 \times 3 \times 3$ Rubik's Cube. On the right side, the changes of the elements after a rotation of the left face (L).

In a concrete example, we can index all of the facets of our puzzle, as it appears on the left side of Figure 1. With this correspondence, we can fix our cube in the Euclidean space and define the order of the faces. This sequence of indices should reflect the same order we use in the case of description for our states. The matrices below can summarize the arrays of the rotations as the elements of a permutation group. We refer to the rotations of the faces with the symbols from the Singmaster notation. Operator L appears in a graphical form on the right side of Figure 1.

$$\mathrm{L} \leftrightarrow \begin{pmatrix} 1 & 4 & 7 & 10 & 13 & 16 & 28 & 29 & 30 & 31 & 33 & 34 & 35 & 36 & 37 & 40 & 43 & 48 & 51 & 54 \\ 10 & 13 & 16 & 37 & 40 & 43 & 30 & 33 & 36 & 29 & 35 & 28 & 31 & 34 & 54 & 51 & 48 & 7 & 4 & 1 \end{pmatrix},$$

$$\mathrm{R} \leftrightarrow \begin{pmatrix} 3 & 6 & 9 & 12 & 15 & 18 & 19 & 20 & 21 & 22 & 24 & 25 & 26 & 27 & 39 & 42 & 45 & 46 & 49 & 52 \\ 52 & 49 & 46 & 3 & 6 & 9 & 21 & 24 & 27 & 20 & 26 & 19 & 22 & 25 & 12 & 15 & 18 & 45 & 42 & 39 \end{pmatrix},$$

$$
\begin{aligned}
\mathrm{U} &\leftrightarrow \begin{pmatrix} 1 & 2 & 3 & 4 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 19 & 20 & 21 & 28 & 29 & 30 & 46 & 47 & 48 \\ 3 & 6 & 9 & 2 & 8 & 1 & 4 & 7 & 28 & 29 & 30 & 10 & 11 & 12 & 46 & 47 & 48 & 19 & 20 & 21 \end{pmatrix}, \\
\mathrm{D} &\leftrightarrow \begin{pmatrix} 16 & 17 & 18 & 25 & 26 & 27 & 34 & 35 & 36 & 37 & 38 & 39 & 40 & 42 & 43 & 44 & 45 & 52 & 53 & 54 \\ 25 & 26 & 27 & 52 & 53 & 54 & 16 & 17 & 18 & 39 & 42 & 45 & 38 & 44 & 37 & 40 & 43 & 34 & 35 & 36 \end{pmatrix}, \\
\mathrm{F} &\leftrightarrow \begin{pmatrix} 7 & 8 & 9 & 10 & 11 & 12 & 13 & 15 & 16 & 17 & 18 & 19 & 22 & 25 & 30 & 33 & 36 & 37 & 38 & 39 \\ 19 & 22 & 25 & 12 & 15 & 18 & 11 & 17 & 10 & 13 & 16 & 39 & 38 & 37 & 9 & 8 & 7 & 30 & 33 & 36 \end{pmatrix}, \\
\mathrm{B} &\leftrightarrow \begin{pmatrix} 1 & 2 & 3 & 21 & 24 & 27 & 28 & 31 & 34 & 43 & 44 & 45 & 46 & 47 & 48 & 49 & 51 & 52 & 53 & 54 \\ 34 & 31 & 28 & 1 & 2 & 3 & 43 & 44 & 45 & 27 & 24 & 21 & 48 & 51 & 54 & 47 & 53 & 46 & 49 & 52 \end{pmatrix}.
\end{aligned}
$$

3.2. Even-numbered Puzzles

When we increase or decrease the number of pieces of our spatial puzzle, we can find some changes compared to the cases where the amount of pieces is an odd number. We can view these differences with the help of the Pocket Cube, where the puzzle arranges its element in a $2 \times 2 \times 2$ layer. In this form, only corner pieces are left. The main novelty of this case is the missing of the middle layer. Without this layer, we can freely rotate any face of the puzzle, but this freedom has a consequence: the fixed orientation of the cube is no longer exists. Without this property, we can not define an order of faces because any face of the puzzle can appear as the front face, right face, or as the other ones in our result. This result is not confusing for a human player because we can easily perceive when we get the shape we expected and finish the game. However, suppose we wanted to store the states in the computer, for example, in the form of a string. In that case, we can generate many invalid sequences of characters just because the whole puzzle has rotated about space, messing up the previously defined order.

In our work, we have forced the fixed position of the $2\times2\times2$ puzzle by fix one of the corner elements. As soon as we eliminate the rotations of L, D, and B, we can designate one of the eight corner pieces, which will not change its position during the game, and it can function as an alternate of the missing central layer. With this modification, we can use the same data storage methods that have appeared in the case of $3 \times 3 \times 3$. In Figure 2, there are the indices of facets in the case of the $2 \times 2 \times 2$ version of Rubik's Cube. Below we introduce a new version of the permutation matrices both for clockwise and counterclockwise quarter-turn rotations. These rearrangements are used in our application to generate a sequence of steps to the solution:

$$
\begin{aligned}
\mathrm{u} &\leftrightarrow \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 \\ 3 & 1 & 4 & 2 & 7 & 8 & 9 & 10 & 11 & 12 & 5 & 6 \end{pmatrix}, \\
\mathrm{r} &\leftrightarrow \begin{pmatrix} 2 & 4 & 8 & 9 & 10 & 11 & 16 & 17 & 18 & 19 & 22 & 24 \\ 8 & 16 & 22 & 17 & 9 & 4 & 24 & 18 & 10 & 2 & 19 & 11 \end{pmatrix}, \\
\mathrm{f} &\leftrightarrow \begin{pmatrix} 3 & 4 & 6 & 7 & 8 & 9 & 14 & 15 & 16 & 17 & 21 & 22 \\ 14 & 6 & 21 & 15 & 7 & 3 & 22 & 16 & 8 & 4 & 17 & 9 \end{pmatrix}, \\
\mathrm{U} &\leftrightarrow \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 \\ 2 & 4 & 1 & 3 & 11 & 12 & 5 & 6 & 7 & 8 & 9 & 10 \end{pmatrix}, \\
\mathrm{R} &\leftrightarrow \begin{pmatrix} 2 & 4 & 8 & 9 & 10 & 11 & 16 & 17 & 18 & 19 & 22 & 24 \\ 19 & 11 & 2 & 10 & 18 & 24 & 4 & 9 & 17 & 22 & 8 & 16 \end{pmatrix}, \\
\mathrm{F} &\leftrightarrow \begin{pmatrix} 3 & 4 & 6 & 7 & 8 & 9 & 14 & 15 & 16 & 17 & 21 & 22 \\ 9 & 17 & 4 & 8 & 16 & 22 & 3 & 7 & 15 & 21 & 6 & 14 \end{pmatrix}.
\end{aligned}
$$

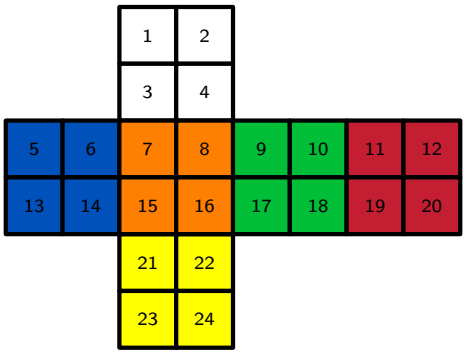

Figure 2. In the case of a spatial puzzle with a size of $2 \times 2 \times 2$ (or another even number), there is no central part, which would fix the orientation of the puzzle in space.

4. Results

In this section, we detail our application and the steps of its development. We describe our starting plans and their results, as well as our attempt to transform some selected models into a spatial puzzle. The gained experiences appear in the final form of our application.

4.1. Functional Requirements and Prototyping

The initial plan for our application was about a game where the player can perform rotations on the different level of layers. To test the feasibility of the plan, we have searched open-source virtual Rubik's Cube implementations.

While we have analyzed these web applications, we also had the opportunity to observe the most important requirements of our puzzle application. For example, where the implementation does not use camera movement, the player cannot check around the puzzle, and only a static viewpoint is allowed during the game. Although it does not make it impossible to find a solution, we can experience this limitation as a significant downgrade compared to the ordinary physical way. Next to the camera movement, any textual information or hint on screen can help us, for example, introducing the previously mentioned Singmaster notation.

During our tests, the GitHub project, called *rubik-js* [14], has been found most useful. This implementation shows the power of animation while the elements are rotating. The code is also a good collection of different Three.js tools. With this one, we could easily explore some of the required elements of the library, for example, the ray tracing object to detect the intersection of our cursor and the pieces of the puzzle.

The project has also served as a basis for our first experiment. Here, we have examined the interaction of the layers of the Rubik's Cube. We have found that, once we can solve any inner layer (a $3 \times 3 \times 3$ in this case) and move up one layer,

we can find a $5 \times 5 \times 5$ layer, which is unsolved. To solve this layer, we have to select a new strategy, which has been developed for this size. After we have solved this layer and go inside to see the changes of the $3 \times 3 \times 3$ layer, we will find that the inner layer has shuffled once again; however, it was solved successfully in a previous step. This fact shows that the solution strategy for the $5 \times 5 \times 5$ version does not pay regard to the inner structure of the puzzle. The relationship of the $5 \times 5 \times 5$ version and the $7 \times 7 \times 7$ cube has the same property.

As a result of this knowledge, the first version of our virtual puzzle has separated the puzzle pieces based on their layers. With the keyboard, the player can define the number of the layer, where the elements should change their position. Every other index is becoming excluded, so the cubes of the inner and outer layers stay at their original place. We also changed the shape of the structure and made many cubes invisible from the original formation of the Rubik's Cube. The resulted pyramid-shaped puzzle can provide a new experience of floating pieces (see Figure 3).

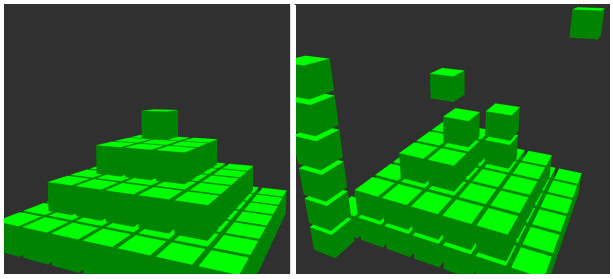

Figure 3. In our first prototype of the application, we found that floating pieces of the puzzle can enrich the possibilities of challenges.

4.2. Solutions to Making Virtual Puzzle Pieces

Our examination of the Rubik's Cube implementations on the web has identified one typical method for puzzle creation: the definition of the same mesh objects in a nested loop. This loop gives an easy operation to generate any size of the Rubik's Cube, depends on the number of its steps.

We can realize that this method leads us to limitations regarded the shapes of the puzzles. Based on the results of [12], we have already seen that these puzzle games work very well in many other cases, even when we turn away from the case of the uniform pieces. If this design principle exists in the physical world, it could work simultaneously in a virtual application. Despite this, during our survey phase, we have not met any existing solution, which presents an arbitrary mesh as the object of the virtual puzzle game in terms of the Rubik's Cube rules.

This idea leads us to that problem, how we can perform any cuts on an arbitrary mesh that results in as many pieces as our dimension of the puzzle requires. For example, following the rules of the $3 \times 3 \times 3$ Rubik's Cube, we have to generate twenty-seven individual puzzle pieces from our mesh.

In our work, we have used the Blender graphical toolset to solve this problem. We have identified two different tools in this software, the *Bisect* tool and the *Boolean* modifier, giving usable results regarded our plans.

The Bisect tool cuts the geometry along a user-defined plane. After this, Blender can consider the cut and help us select a region of faces. At the end of this process, the vertices of the active area can split off from the rest of the mesh. In our work, we avoid the usage of this tool because of the experience of the following disadvantages:

- The cut of complex meshes can be challenging. In these cases, the defined plane cuts many triangles of the mesh, but Blender can not handle this separation of the vertices efficiently. In some cases, the active area contains vertices from the wrong side of the defined plane. The correction of this error requires minor manipulations of selection by hand, which expand the preparation time.
- When we split off the active area from the other part of the mesh, the new results have a missing face at the point of separation. The creation of new faces inserts one plus step to our process.

These drawbacks do not exist in the case of the Boolean modifier. Here, the intersection operator from set theory can help us to generate the puzzle pieces. For this, we can draw cube meshes around our model, and we can apply the intersection operator of the modifier for every cube and the loaded mesh. The snapshots of this process are in Figure 4. We can see that the remaining parts of the cubes result in the shape of our original loaded mesh, but now its pieces are individual ones. Accordingly, the original mesh can be deleted from our scene at the end of the process. The usage of this tool looks promising as a basis of an automated procedure for puzzle pieces creation. Nevertheless, before we do this, we should examine the scaling of models in more detail to find the best-joined scale for both the mesh and the cubes as helpers.

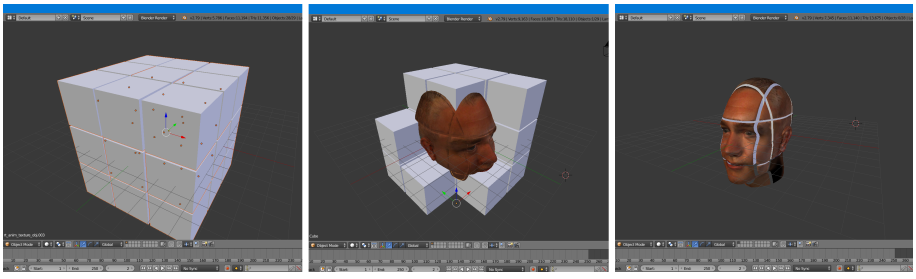

Figure 4. The steps of the puzzle preparation process in Blender with the Boolean modifier. First, we create several cubes and arrange them to look similar to the Rubik's Cube. After this, we can use the Boolean modifier to remove unnecessary parts of this cube mesh. At the end of the process, the resulting shape is the same as the shape of the initially loaded mesh.

We can use different data formats to export these puzzles from Blender, for example, the *obj* format or the XML-based *Collada* format. Then, the Three.js library provides high-level methods to process this graphical data in our application.

4.3. Application Overview

The final form of our application uses the models imported from Blender as puzzles. The display of the application has two parts, the central part is the scene for the puzzle, and there is a navigation bar on the top of the screen with buttons to provide additional features (see Figure 5).

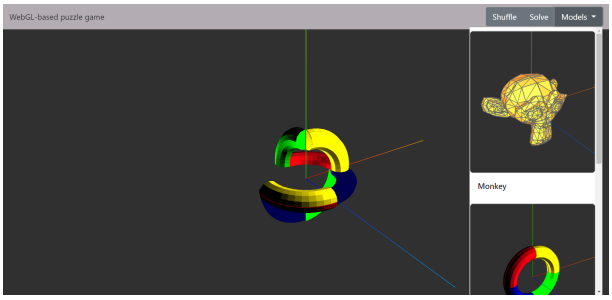

Figure 5. The display of the application and its main features.

The player can find an axis helper object on the screen also, provided by the Three.js library. This object helps to navigate in the scene, and it helps to recognize the directions after several rotations have been performed or the view has changed.

During the game, the user can select from different models. On the navigation bar, the players can open a dropdown list with the *Models* button, and they can click on the card of the puzzle. Then the display refreshes, and the new model will be rendered on the screen. There are three predefined models in the current version of the application. Two of these puzzles have the size $2 \times 2 \times 2$, and there is one $3 \times 3 \times 3$ version also, which provides a variety of challenges for the player (see Figure 6).

The user can shuffle the selected puzzle with cursor movements, which identify the group of the nearest elements and the direction of the movement. We can trigger these actions by calling related JavaScript events, for example, the *onmousedown* event, when the user presses a mouse button. After this, the methods of spatial rotations about axes are well defined in the work of [14] as well as in the functions from the Three.js library. However, in some cases, we can recognize unexpected movements when these tools can not calculate z-direction correctly, and they identify x or y as the selected axis.

To help the player, we have implemented a solution algorithm for the application, which is available from the navigation bar. This strategy can generate steps to solve any permutation of the $2 \times 2 \times 2$ puzzle. This implemented algorithm can not solve the $3 \times 3 \times 3$ version. We indicate the missing of the automatic solver

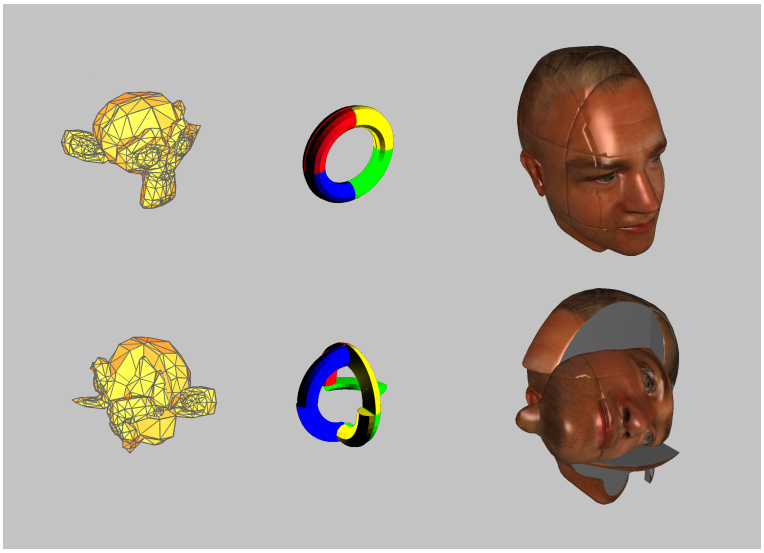

Figure 6. The three puzzles of our WebGL application. We have performed the rotation sequence rUF on each puzzle. On the left side, we can see the model with applying basic flat shading and displaying the surface lines. The torus mesh uses the Phong material of the Three.js library. In this case, we have applied colors to identify the pieces of this puzzle. The puzzle of a virtual male head has flat shading again, but this time we have applied textures, which improves perception even more compared to the first case.

for this version by removing of the *Solve* button from the navigation bar while the user is playing with the $3 \times 3 \times 3$ puzzle. In other cases, the button appears on the screen, and the user can use it whenever the game seems complicated.

5. Conclusion

In this paper, we have introduced a WebGL-based application for skill developing purposes. Our application includes spatial puzzles for this aim. Before we have detailed the mechanics of the spatial puzzle games, we have investigated the existing tools and technologies, which could help us to realize the feasibility of our plan.

We have studied and introduced the theoretical foundations of spatial puzzles with the related works about the well-known spatial puzzle toy, the Rubik's Cube. The mathematical properties of this puzzle have helped us implement a solution algorithm for our application to help users.

Next to this, we have searched the possibilities of new shapes for our spatial puzzle game. In this area, we have surveyed some existing Rubik's Cube implementations, and we have described a related research result of the physical design

of puzzles. We have found that the experienced limitations by our physical reality can be reduced or eliminated in the virtual environment. Based on this knowledge, we have used the Blender graphical toolset, and with this software, we have created spatial puzzle pieces from arbitrary meshes for our application.

6. Future Work

We plan to implement more solution algorithms to help the user while playing with every kind of puzzle with a general size of $n \times n \times n$. To create these general-sized puzzles, we plan to implement algorithms to support users to create puzzle pieces from any mesh automatically.

We would like to reduce the identified disadvantages of the user experience in the future version of our application, for example, the missing of the tactile experience. To do this, we would like to extend our solution with tools from VR and AR technologies.

References

[1] *Blender v2.79. (software)*, The Blender Foundation, Available online: `https://www.blender.org/` Accessed: 2020-01-16.

[2] R. Cabello: *three.js–JavaScript 3D library (r112) (software)*, Available online: `https://threejs.org/` Accessed: 2020-02-20.

[3] R. Caillois: *The Structure and Classification of Games*, Diogenes 3.12 (1955), pp. 62–75, doi: `https://doi.org/10.1177/039219215500301204`.

[4] A. Evans, M. Romeo, A. Bahrehmand, J. Agenjo, J. Blat: *3D graphics on the web: A survey*, Computers & Graphics 41 (2014), pp. 43–61, doi: `https://doi.org/10.1016/j.cag.2014.02.002`.

[5] H. Fitriyani, N. Tasu'ah: *The Use of Three Dimensional Puzzle as a Media to Improve Visual-Spatial Intelligence of Children Aged 5-6 Years Old*, Indonesian Journal of Early Childhood Education Studies 3.1 (2014), `https://journal.unnes.ac.id/sju/index.php/ijeces/article/view/9476`, pp. 74–78, doi: `https://doi.org/10.15294/ijeces.v3i1.9476`.

[6] M. Gymrek, J. Li, E. Demaine: *The Mathematics of the Rubik's Cube, Introduction to Group Theory and Permutation Puzzles*, SP.268 Coursenotes, Massachusetts Institute of Technology, USA, Spring 2011, Available online: `https://web.mit.edu/sp.268/www/rubik.pdf` Accessed: 2019-12-21.

[7] N. Hering, M. Rünz, L. Sarnecki, L. Priese: *3DCIS: A Real-time Browser-rendered 3D Campus Information System Based On WebGL*, Proceedings of the 2011 International Conference on Modeling, Simulation & Visualization Methods, MSV 2011 (2011), `http://worldcomp-proceedings.com/proc/p2011/MSV3273.pdf`, pp. 10–15.

[8] D. Joyner: *Adventures in Group Theory - Rubik's Cube, Merlin's Machine and Other Mathematical Toys, 2nd Edition*, ISBN: 9780801890130, Baltimore, Maryland: The Johns Hopkins University Press., 2008.

[9] S. Kim: *From Physical Game to Computer Game*, CS.215 Coursenotes, Wellesley College, Massachusetts, USA, Fall 2014, Available online: `http://cs.wellesley.edu/~cs215/Lectures/L17-IntroGamesJigsawPuzzle/From\%20Physical\%20to\%20Computer\%20Game.pdf` Accessed: 2020-02-02.

[10] M. Králik, K. Žáková: *Interactive WebGL Model of Hydraulic Plant*, IFAC-PapersOnLine 48.29 (2015), pp. 146–151,
doi: https://doi.org/10.1016/j.ifacol.2015.11.228.

[11] I. Papp, R. Tornai, M. Zichar: *What 3D technologies can bring to education: The impacts of acquiring a 3D printer*, 2016 7th IEEE International Conference on Cognitive Infocommunications (CogInfoCom) (2016), pp. 000257–000262,
doi: https://doi.org/10.1109/CogInfoCom.2016.7804558.

[12] T. Sun, C. Zheng: *Computational Design of Twisty Joints and Puzzles*, ACM Transactions on Graphics 34.4 (2015), 101:1–101:11,
doi: https://doi.org/10.1145/2766961.

[13] R. Tóth, M. Zichar, M. Hoffmann: *Improving and Measuring Spatial Skills with Augmented Reality and Gamification*, in: ICGG 2020 - Proceedings of the 19th International Conference on Geometry and Graphics, ed. by L.-Y. Cheng, Cham: Springer International Publishing, 2021, pp. 755–764,
doi: https://doi.org/10.1007/978-3-030-63403-2_68.

[14] J. Whitfield-Seed: *rubik-js (software)*, 2013, Available online: https://github.com/joews/rubik-js Accessed: 2020-02-21.

[15] A. Yuniarti, A. Atminanto, A. Mardasatria, R. R. Hariadi, N. Suciati: *3D ITS campus on the web: A WebGL implementation*, 2015 International Conference on Information & Communication Technology and Systems (ICTS) (2015), pp. 141–144,
doi: https://doi.org/10.1109/ICTS.2015.7379888.

[16] P. G. Zimbardo, R. L. Johnson, V. McCann: *Psychology: Core Concepts, 8th Edition*, ISBN-13: 9780134190839, New York, New York: Pearson, 2017.