Game-based Transformations: A playful approach to learning transformations in computer graphics

Martin Eisemann[1]

[1]TU Braunschweig, Germany

Abstract

In this paper, we present a playful and game-based learning approach to teaching transformations in a second-year undergraduate computer graphics course. While the theoretical concepts were taught in class, the exercise consists of two web-based tools that help the students to get a playful grasp on the complex topic, which is the foundation for many of the later concepts typically taught in computer graphics, such as the rendering pipeline, animation, camera motion, shadow mapping and many more. The final students' projects and feedback indicate that the game-based introduction was well-received by the students.

CCS Concepts

• ***Computing methodologies*** $\rightarrow$ ***Computer graphics;*** • ***Applied computing*** $\rightarrow$ ***Interactive learning environments;***

1. Introduction

Transformations are an essential part of every computer graphics course that builds the foundation for most of the advanced topics in this area. Therefore, understanding transformations is essential for students. Unfortunately, many factors discourage students, especially undergraduates, from learning this topic properly. Firstly, understanding and applying the concepts require a solid mathematical foundation, and especially the matrix transformations can be overwhelming. Secondly, the provided coding frameworks can be intimidating and distracting for students, who then often struggle to focus on the task at hand. Thirdly, in most frameworks, there is hardly any feedback, and the impact of code changes is sometimes hard to comprehend. Especially in graphics, errors often result in a disappearance of the rendered content with few possibilities for proper debugging.

To counter these negative effects a game-based and playful approach was proposed to 2nd year undergraduate computer science students in a computer graphics and animation course at a German university of applied science. While the theoretical background was taught in a 90-minute class, the respective practical labs consisted of two parts: A web-based scene graph tool for an interactive playful introduction to transformations and a physics-based matrix browser game with coding functionality that focuses on applying the transformations. In a final optional project at the end of the course the students could show that they know how to apply the learned concepts by building a self-made rendering engine from scratch.

The overall goal of our approach is to improve how the tasks are perceived emotionally by the students, diminishing frustration and increasing positive emotions, as these are crucial components for successful learning [PLM*17, GH13].

In particular we wanted to address the following goals:

1. Students should be enabled to conduct the practical lab without the hassle of setting up and compiling a complex framework;
2. The focus should lie on the concepts and no coding framework should distract the students;
3. The students should be able to interactively investigate the different concepts. In this case the basic transformations including translation, rotation around the coordinate axes, scaling, scene graphs, and multiplication of transformation matrices to grasp the concept of dependent transformations;
4. The students should get immediate feedback on their actions;
5. The students should get both a high- and low-level understanding of the concepts of transformations;
6. A gentle increase in the difficulty level going from an interactive playful exploration to a game-based coding environment to a full-code rendering engine should keep frustration to a minimum;
7. The intrinsic motivation of the students should be improved by providing a game-based learning environment.

Feedback by the students was entirely positive, and especially students with lower mathematics and programming skills appreciated the slower transition from an interactive tool to actual coding. The topic of transformations was a 1-week part of a 15-weeks, 5 ECTS credit course on computer graphics with focus on rasterization and real-time rendering.

DOI: 10.2312/eged.20231018

2. Overview

In the following, we will describe our approach to teaching transformations to students in a playful and non-frustrating way, which increases their intrinsic motivation to learn the concepts and mathematical backgrounds required to understand the topic.

We will start by reviewing related work (Sec. 3) and giving an overview of the course (Sec. 4). Next, we will discuss the originality of our approach (Sec. 5). After that we will describe our assignments and tools in detail (Sec. 6). Finally, we will discuss our approach (Sec. 7) and draw a conclusion (Sec. 8).

3. Related Work

Teaching computer graphics has always been and still is a challenging task. It requires applying mathematical concepts in a difficult programming environment (including application programming and shader programming) which is often tough to debug. Some approaches start from scratch to give the student full control over their render engine [CXR18], while others build on existing gaming platforms [Los22, AG16] or adapt WebGL and high-level frameworks like ThreeJS for an easier start [AG16]. All of these have in common that students start with a basic framework that they extend throughout the course. However, few consider specialized or gaming-based assignments to help the students to playfully engage with a certain topic. Gaming-based assignments have been successfully used in other areas as well, such as teaching programming [AM07, MTVJ10].

In addition, it has been shown, that though students are often reluctant to employ math, they show interest in computer graphics [She15]. We want to make use of this fact by minimizing the frustration level of students. We exemplify this by describing our approach to teaching transformations, one of the key topics in graphics [BWF17]. We were inspired by Nate Robins' classic OpenGL tutorials [Rob00], which provide a variety of small programs to interactively explore different aspects of the rendering pipeline. Another inspiration comes from the I3T tool [FMF*18], which allows you to a series of transformation matrices, change their entries and apply the result to a simple 3D object. Our approach extends this to a more playful setting with increasing complexity. In our Scene Graph Tool, students have a more realistic scenario with a visual representation of the graph to see the dependencies. They can interactively change the values in the matrices using sliders. The tool has a consistent color coding to see how all the values are related. The Matrix Game extends this to a game-based learning scenario, which includes real programming tasks for a gentler transition to the final code project.

4. Course Overview

Our approach to teaching transformations was tested in a 5 ECTS compulsory course for computer graphics and animation in the second year of a bachelor's degree in computer science at TH Köln - University of Applied Sciences. The students had no previous knowledge of computer graphics and only limited mathematical skills. Some had never even heard of matrices before taking the course because they began their studies during the summer semester and therefore missed the linear algebra course before taking part in the computer graphics course. 22 students, aged between 19 and 27, took part in this course, which had the following learning objectives:

- Understand the typical algorithms used in real-time computer graphics and know how to implement them;
- Implement highly parallel algorithms on the GPU using OpenGL and GLSL;
- Apply the gained knowledge of the rendering pipeline, transformations, light, materials, and textures to produce a small real-time rendering engine or game.

As the students were taught Java during their first two semesters, we used it as well, and used GLSL for shader programming.

The overall course syllabus was as follows:

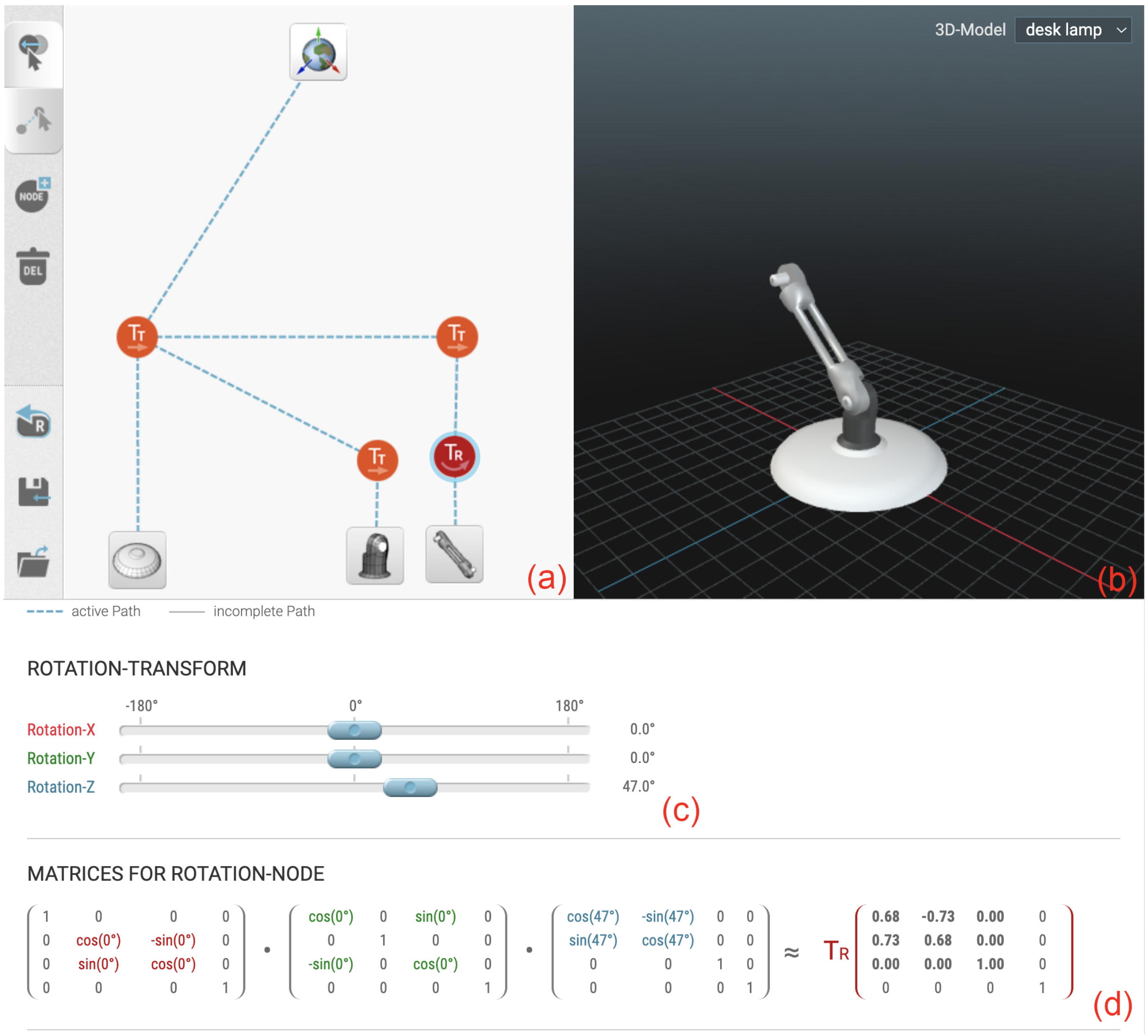
1. History of computer graphics
2. Math basics
3. Raster- and Vectorgraphics
4. Scene representation and scene primitives
5. Introduction to OpenGL and the rendering pipeline
6. Introduction to GLSL and the programmable graphics pipeline
7. **Basic Transformations, including rotation, translation, scaling, projective representations, and matrix representations**
8. **Scene Graph**
9. Textures
10. Camera models (view transformation, normalized device space, orthographic and perspective projection)
11. Material and lighting
12. Advanced texturing
13. Ray Tracing
14. Deferred shading and shadows

Afterward, an optional rendering competition took place where students could show how they apply their knowledge to write a small render engine/game within three weeks, similar in spirit to [PBB19].

The course content was provided in classic 90-minute lectures, which were also provided as videos on the course website [ADLR17]. Weekly programming assignments had to be conducted in small teams of 3-4 people. The course finished with an exam and an optional rendering competition that was not graded. That is, the course followed a classic teaching approach, and only the transformations were taught using the tools described in this paper and following the goals stated in Sec. 1. The web-based tools also helped to switch from on-site to online teaching during the COVID-19 pandemic, where the course was taught using a video conference tool (Zoom) to hold the lectures and conduct labs.

5. Originality and Teaching Approach

To fulfill our objectives from Sec. 1, we evaluated our approach to teaching transformations in an undergraduate computer graphics and animation course. We taught the theory in a 90-minute lecture to explain ideas and basic concepts of transformations and how to apply them to place and animate rigid objects. A special focus was on the typical transformations: rotation, translation, and scaling,



ROTATION-TRANSFORM

	-180°	0°	180°	
Rotation-X				0.0°
Rotation-Y				0.0°
Rotation-Z				47.0°

MATRICES FOR ROTATION-NODE

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(0°) & -\sin(0°) & 0 \\ 0 & \sin(0°) & \cos(0°) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} \cos(0°) & 0 & \sin(0°) & 0 \\ 0 & 1 & 0 & 0 \\ -\sin(0°) & 0 & \cos(0°) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} \cos(47°) & -\sin(47°) & 0 & 0 \\ \sin(47°) & \cos(47°) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \approx T_R \begin{pmatrix} 0.68 & -0.73 & 0.00 & 0 \\ 0.73 & 0.68 & 0.00 & 0 \\ 0.00 & 0.00 & 1.00 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Figure 1: *Main screen of the scene graph tool. (a) Scene graph, (b) 3D view, (c) interactive sliders to change the selected transformation node, (d) transformation matrix of the selected node.*

and their representations as 4×4 matrices, as well as how to combine them into more complex transformations and scene graphs.

The originality stems from the way the concepts were examined by the students in the practical labs. Instead of letting the students extend the typical graphics framework right up front, we chose a gentle and more motivating approach. In the first task, the students had to playfully investigate transformations by examining our web-based scene graph tool (Sec. 6.1). In the second task, the students could apply the gained knowledge in a physics-based browser game (Sec. 6.2), where the task was to write small code snippets to create transformation matrices to place wooden blocks in a 3D environment such that a marble is guided towards a certain goal. Finally, the students had to write their own graphics application in small teams at the end of the course, which was optional and not graded but a certificate was handed out to the best team (Sec. 6.3).

6. Assignment and Tools

The assignment for the lecture on transformations was two-fold. Firstly, the students had to load our Scene Graph Tool in their browser. The task was to build a scene graph to correctly animate a lamp. Optionally, the more difficult task of animating a robot hand could be conducted. Secondly, the students had to load the website of our matrix game and solve the first three puzzles, which increase in complexity. The last three were optional.

6.1. Scene Graph Tool

Figure 1 shows an overview of the provided scene graph tool. The GUI consists of (a) the scene graph including the transformation nodes (red and orange), geometry nodes (icons depicting the 3D geometry) and a root node (showing a world with coordinate axes

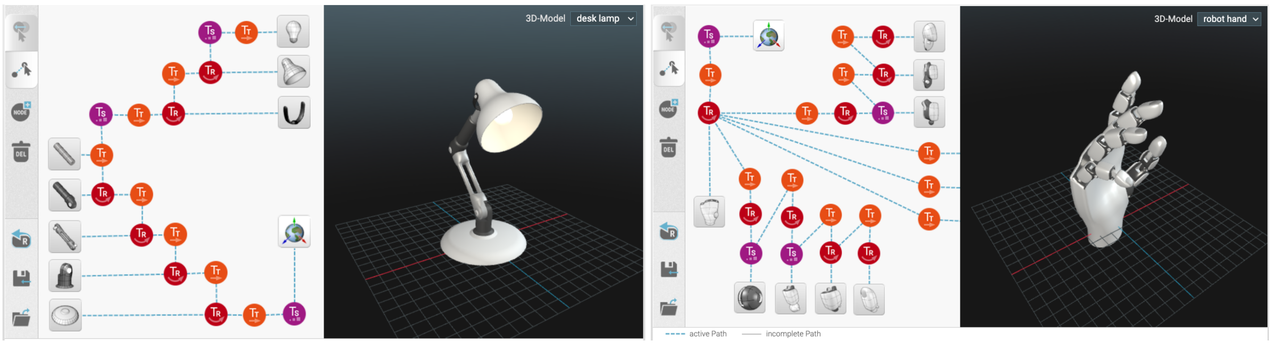


Figure 2: *Potential solution to the two provided scenes.*

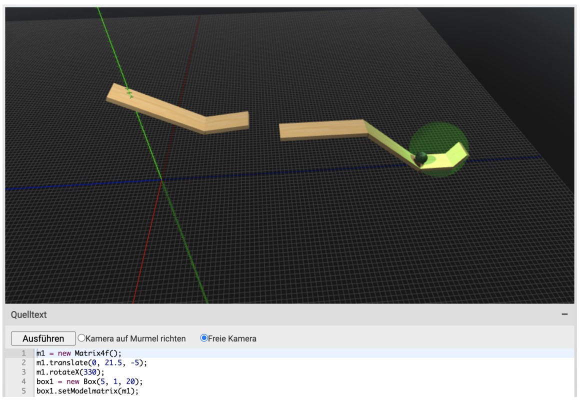
to highlight that the goal is to transform a geometry node from object space into world space). The left panel provides functionalities to add new nodes, move nodes, change connections in the scene graph, delete nodes and connections and to import and export the solution (needed for the grading of the assignment and to save intermediate results). The scene graph gets rendered by collecting the transformation matrices from a geometry node to the root node and applying the resulting matrix to the geometry as model matrix.

A 3D visualization of the current scene (b) is updated after every change to the scene graph. The user can freely navigate around the object and inspect the impact of changes to the scene graph.

The lower half of the GUI consists of two parts. In (c), we have interactive sliders to change the values for the current transformation node. If a translation node is selected, the sliders can move the object along the three coordinate axes. If a scaling node is selected, these are for scaling the object along the coordinate axes, and if a rotation node is selected, the sliders can be used to rotate the affected object around the x, y, or z-axis. If the sliders are moved, the students can immediately see the effect on the 3D visualization. As the camera is initialy focusing on the center of the coordinate system and all matrices are initialized to the identity matrix, there is no danger of creating objects outside the view frustum. If the geometry nodes are not connected properly to the root node they are rendered in their respective object coordinate system. All objects have been designed to have their midpoint at the origin and scaled so that they fit in the view frustum.

Underneath the sliders (d), the corresponding matrices are displayed. This helps the students get used to the matrix representations and how they are applied to transform the objects. Each transformation from the sliders is represented as a single matrix that gets multiplied to reveal the final transformation matrix of the node on the bottom right.

We make strong use of color coding to highlight the semantic connection between the different components. The three sliders are marked in red, green, and blue, corresponding to the color of the coordinate axes, which are displayed in the 3D viewer, and also corresponding to the respective entries in the matrices. The rendered view and matrices get updated in real-time whenever the user makes changes to the sliders or scene graph. This way, the student gets immediate feedback on their changes and a playful environment is created. Within this they can experiment without the fear of breaking the system, which gives them a gentle introduction to the matrix representation.



Ausführen [ ] Kamera auf Murmel richten [✓ Freie Kamera]

```
m1 = new Matrix4f();
m1.translate(0, 21.5, -5);
m1.rotateX(330);
box1 = new Box(5, 1, 20);
box1.setModelmatrix(m1);
```

Figure 3: *Main screen of the Matrix Game. Top: The freely movable 3D view, bottom: The code window to place the wooden bricks.*

The task is now to build a scene graph that correctly animates the 3D object when the values of the different nodes are changed. For example, in Fig. 1, the joints are moved with the ground plate as their respective nodes in the scene graph are linked with the transformation node of the ground plate. In our tool, we offer two models to play with: A lamp consisting of eight objects, and a robot hand consisting of 29 objects, though we do not expect the students to build the full hand. Fig. 2 shows a potential solution for both scenes.

The tool is implemented using HTML, JavaScript, and Three.js, a high-level JavaScript framework for WebGL applications.

6.2. The Matrix Game

Once the students get used to the Scene Graph Tool, they can continue with the next task, the Matrix Game, Fig. 3. The Matrix Game is a physics-based simulation game. The goal is to guide a marble towards a marked goal and have it stay there for three seconds. The user's task is to write Java code to create matrices to transform wooden boxes so that the marble can reach its goal. The user is

```
m1 = new Matrix4f();
m1.translate(0,35,0);
m1.rotateX(-33);
box1 = new Box(5,1,15);
box1.setModelmatrix(m1);
```

Figure 4: *Example code to transform one box in the Matrix Game. The code needs to be written by the students.*

provided with several convenience functions that are in alignment with the framework used throughout the rest of the course. An example of such code is given in Fig. 4. In particular, the user can create a box of a certain size, centered around the origin of the coordinate system using `new Box(width, height, depth)`. Additionally, they can create a matrix (`new Matrix4f()`) and apply transformations to them. Finally, the user can set the model matrix of a box to a certain matrix which is then applied during rendering. Underneath the code editor, the students have a short manual that describes the interaction with the tool and a command reference. That way, the students can practice their coding skills in a safe and playful environment. The game consists of six levels with increasing difficulty. Fig. 3 shows the introductory level which is used as a playground to explain the concept. Fig. 5 shows the third level, where the marble is blocked by a concrete block and level 6, which is only for real experts as the marble has to go through a maze. If a student can solve this, one can safely assume that they are proficient in dealing with model matrix transformations. Level 2 is simply an empty level, in level 4, the marble has to go around a wall, and in level 5, the goal is placed diagonally of the starting position blocked by two concrete walls. In the assignment, we asked the students to solve the first three levels, but many chose to continue and solve the others as well.

The tool is implemented using HTML, JavaScript, PHP, and Three.js. The PHP parser converts the written code in the editor into executable JavaScript code using regular expressions. Transformations are internally saved in text format before converting them to Three.js matrices when calling the *setModelMatrix*-function. The physics are implemented using Physijs.

6.3. Final project

At the end of the course, the students were asked to implement a small 3D scene or game using the course framework. This framework was extended during the course but provides only basic OpenGL functionalities, like setting up a window, loading 3D objects and textures, or setting up shaders and uploading matrices. The programming language was Java, as this was taught during the students' first year, in conjunction with the LWJGL library for the OpenGL functionality.

About 50% of the students took part in this optional project, even though they did not receive any credits for it. Some results are shown in Fig. 6. We were positively surprised to see that some of the projects were real games that could be played and involved complex topics such as character animation. Videos are shown in the supplemental material. All projects showed that the transformation concepts taught could be applied in real-world scenarios by the students, which was the main goal of the assignment.

7. Discussion

All students who successfully finished the assignments and took part in the final project completed the course successfully. The students were highly motivated, and even though some complained that the final project was too much work, they still took part in the contest.

The provided Scene Graph tool and Matrix Game were well received, and we got much positive feedback that it helped to learn the concepts without frustration. The teaching evaluation revealed that the students especially loved the didactic structure and the feedback for self-assessment provided by the tools (grading both with 1.2 where 1 is the best and 5 the worst grade). The amount of work that had to be conducted was perceived as average, so the workload was en par with other courses.

Additionally, the proposed structure to teach transformations follows the *Four I's Recipe* for designing computer graphics exercises [PA14]. The tools are independent of other assignments; incremental in the sense that the tasks get more and more complex (interactive scene graph to coding transformations in the matrix game to the final project); the activities are iterative (different levels in the matrix game and the concepts are used throughout the remaining assignments of the course); and integrative by offering the final project where all learned concepts are to be applied. Therefore, we conclude that using our teaching approach, we could achieve the goals stated in Sec. 1.

8. Conclusion

This work proposed an experimental new way to teach computer graphics. While we focused on teaching transformations in this paper, we see strong evidence that the chosen approach of gently and playfully introducing students to a topic can help to diminish frustration and foster intrinsic motivation instead. Other topics like camera transformation, texture mapping, shader programming, and others which would all benefit from such an approach. The results indicate that the course was well perceived and helped the students to gain insight into the complex topic of computer graphics. The tools will be available from this paper's website at https://graphics.tu-bs.de/publications/eisemann2023game-based.

9. Acknowledgment

Special thanks goes to Jarek Sarbiewski and to Andreas Pahlen and Stefan Förster for doing such a wonderful job on implementing the Scene Graph Tool and Matrix Game, respectively.

References

[ADLR17] Anderson E., Duchowski A., Liarokapis F., Redford A.: The new CGEMS - preparing the computer graphics educational materials source to meet the needs of educators. In *Eurographics 2017 - Education Papers* (2017), Eurographics Association. 2

[AG16] Amador G., Gomes A.: A video games technologies course: Teaching, learning, and research. In *Eurographics 2016 - Education Papers* (2016), Eurographics Association. 2

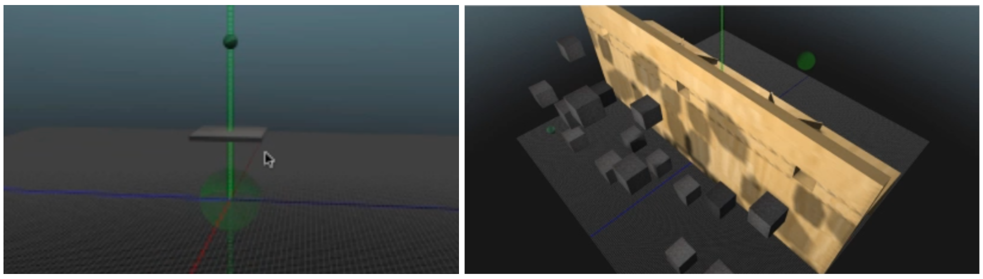


Figure 5: *Third level with intermediate difficulty and sixth level for experts.*

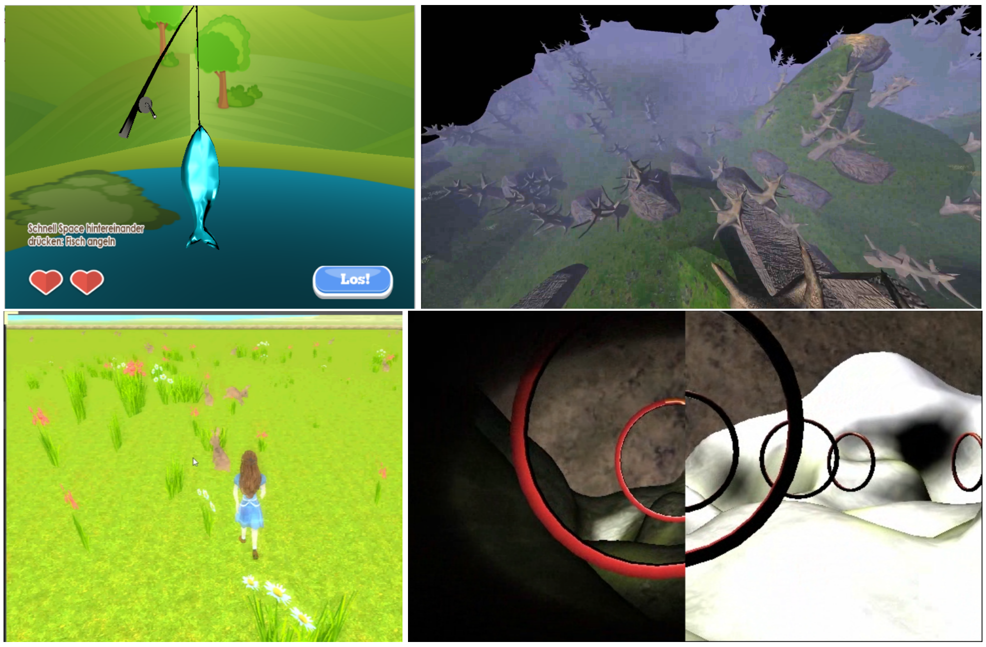


Figure 6: *Examples of the final students' projects.*

[AM07] Anderson E. F., McLoughlin L.: Critters in the classroom: A 3d computer-game-like tool for teaching programming to computer animation students. In *ACM SIGGRAPH 2007 Educators Program* (2007). 2

[BWF17] Balreira D., Walter M., Fellner D.: What we are teaching in introduction to computer graphics. In *Eurographics 2017 - Education Papers* (2017), Eurographics Association, p. 1–7. 2

[CXR18] Chen M., Xu Z., Rippin W.: On the pedagogy of teaching introductory computer graphics without rendering apis. In *Eurographics 2018 - Education Papers* (2018), Eurographics Association. 2

[FMF*18] Felkel P., Magana A. J., Folta M., Sears A. G., Benes B.: I3T: Using Interactive Computer Graphics to Teach Geometric Transformations. In *Eurographics 2018 - Education Papers* (2018), Post F., Žára J., (Eds.), The Eurographics Association. doi:10.2312/eged.20181000. 2

[GH13] Goetz T., Hall N.: Emotion and achievement in the classroom. In *International guide to student achievement* (2013), p. 192–195. 1

[Los22] Loscos C.: Introduction to computer graphics: a visual interactive approach. In *Eurographics 2022 - Education Papers* (2022), Eurographics Association. 2

[MTVJ10] Muratet M., Torguet P., Viallet F., Jessel J.-P.: Experimental Feedback on Prog and Play, a Serious Game for Programming Practice. In *Eurographics 2010 - Education Papers* (2010), The Eurographics Association. 2

[PA14] Peters C. E., Anderson E. F.: The Four I's Recipe for Cooking Up Computer Graphics Exercises and Assessments. In *Eurographics 2014 - Education Papers* (2014), The Eurographics Association. 5

[PBB19] Palus J.-P., Belhadj F., Bourdin J.-J.: Do contests improve students skills in Computer Graphics? The case of API8. In *Eurographics 2019 - Education Papers* (2019), Tarini M., Galin E., (Eds.), The Eurographics Association. 2

[PLM*17] Pekrun R., Lichtenfeld S., Marsh H., Murayama K., Goetz T.: Achievement emotions and academic performance: Longitudinal models of reciprocal effects. *Child Development 88*, 5 (2017). 1

[Rob00] Robins N.: Nate robins' opengl tutorials, 2000. http://user.xmission.com/ nate/tutors.html. 2

[She15] Shesh A.: Teaching graphics to students struggling in math: An experience. In *Eurographics 2015 - Education Papers* (2015), Eurographics Association, p. 23–29. 2