UNIVERSITY OF TARTU
FACULTY OF MATHEMATICS AND COMPUTER SCIENCE
Institute of Computer Science
Computer Science

Raimond-Hendrik Tunnel

Computer Graphics Learning Materials

Master's Thesis (30 ECTS)

Supervisors: Konstantin Tretyakov, MSc
Anne Villems, MSc

TARTU 2015

Computer Graphics Learning Materials

Abstract:

This thesis provides an overview of the learning material and a custom learning environment created for the Computer Graphics (MTAT.03.015) course in the University of Tartu. It describes a modular layout, that mixes a top-down and bottom-up approaches, in which the course was organized. The created material also includes interactive examples that satisfy engagement level 4 requirements. The specification and implementation details of the custom learning environment called CGLearn are given. Thesis concludes with the analysis of the feedback questionnaire answered by the students participating in the course and using the material.

Keywords

Computer graphics, teaching, education, learning environment, interactive examples.

Arvutigraafika õppematerjal

Lühikokkuvõte:

Selles lõputöös on antud ülevaade Tartu Ülikooli aine Arvutigraafika (MTAT.03.015) jaoks koostatud õppematerjalist ja õppekeskkonnast. Kirjeldatud on aine modulaarset ülesehitust, mis rakendab kombineeritud ülevalt-alla (ing. k. *top-down*) ja alt-üles (ing. k. *bottom-up)* lähenemisi. Loodud õppematerjal sisaldab endas interaktiivseid näiteid, mis vastavad hõivatuse taksonoomia 4ndale tasemele. Õppekeskkonna CGLearn spetsifikatsioon ja implementatsiooni detailid on kirjeldatud. Töö lõpus on kursusel osalenud õpilaste hulgas läbi viidud tagasiside küsitluse tulemuste analüüsiga.

Võtmesõnad:

Arvutigraafika, õpetamine, haridus, õppekeskkond, interaktiivsed näited.

Table of Contents

1. Introduction

Computer graphics is a computer science subject taught in both undergraduate and graduate levels in different universities ([1] and [2]). In the University of Tartu, Institute of Computer Science, the Computer Graphics (MTAT.03.015) course has been an elective 6 ECTS credits course in different Master curricula [3] from 2002/2003 to 2007/2008. In reality, the course has been conducted only as an optional course on the fall of 2005 and 2013, and spring of 2015. Latter was done for the work of this thesis. The previous and current layouts, learning activities, and the grading system of the course are described in detail in chapter 2.

When studying computer graphics, it is important that the students get an understanding of different graphics algorithms. Facilitating the understanding and skills effectively is a subject of much debate. For computer graphics, educators have proposed interactive examples to demonstrate different algorithms to students and have them experiment with their parameters. One such way is proposed by Naiman in [4]. That included a collection of interactive examples, called teaching modules, and a library routine for creating them. Authors conclude that the primary benefit for the students was, that instead of relying only on static text and imagery, they were exposed to dynamic material, that increased interest, attention and comprehension. The modules by Naiman, were implemented in C and thus were suitable for demonstrating the concepts in a computer lab, during practice sessions.

With the growing popularity of the World Wide Web, Klein *et al* proposed a web-based collection of such interactive examples in [5]. Authors state that computer graphics topics could not be adequately presented with traditional education methods at the time. Their solutions consisted of HTML pages and interactive Java applets. The conclusion was that this approach allowed students to prepare, catch-up and deepen the understanding of the subjects without the need to install specific software.

Another approach with an e-learning focus by Capay and Tomanova [6] proposed interactive examples implemented in Adobe Flash. They used the examples mainly during the weekly lessons, but also noted the possibility of students interacting with them during individual study. Authors describe their use of the learning management system Moodle [7] in their computer graphics course.

All of those proposed examples had aspects that made them not entirely suitable for the current Computer Graphics course. The C library would have restricted students to only interact with the examples in a computer lab, without having them to compile the examples for their own platforms. Java and Flash applets proposed by the other two articles had an intrinsic problem of relying on third-party plugins to run in a browser. During the last two decades, we have seen the decline of both technologies on the web. Although at the time when those approaches were proposed there was not much of an alternative for graphics.

During the work on this thesis, a specialized online learning environment called CGLearn (https://cglearn.codelight.eu) was created. The environment enables students to read the material of the course, interact with different computer graphics examples, submit solutions to practice tasks, facilitates communication relating to the feedback on those solutions, and enables them to see their results in comparison to the statistical representation of overall results. CGLearn's back end is implemented in PHP and front end (together with the interactive examples) uses JavaScript. The specific requirements and the implementation is described in chapter 3.

Previously the Computer Graphics course did not have online material available to the students, besides the lecture slides and practice session work-sheets [8]. There were references to material of similar computer graphics courses in other universities, other online material related to computer graphics, and a number of text books. During the work of this thesis, a collection of materials was written in CGLearn that approximates the layout of the topics covered during the course. Material includes interactive examples accompanying many of the algorithms and concepts described there. The description of the material and the interactive examples is given in chapter 4.

CGLearn also includes practical tasks for the students. The tasks include a description of an algorithm or a technique, the expected outcome and a base-code. Solutions could be presented by students in both JavaScript or C++. The nature of the tasks and their overview is given in chapter 5.

In order to allow students to test themselves on the material, an implementation of flashcards was added to CGLearn. Flashcards are based on the SuperMemo 2 algorithm [9] and are described more in chapter 6.

Assessment of the CGLearn learning environment and the course material was made in a form of a feedback questionnaire. Because at the time of writing this thesis the course was still ongoing, the results reflect mostly the students' experiences so far. The questionnaire consists of questions about the course organization, lectures, practice session tasks, material and functionality in CGLearn.

Lectures were additionally assessed by analyzing exit cards. These are cards, where students had to write an answer to two general questions about what they learned during the lecture, and what else would interest them. This technique is further explained in the thesis. The results of both the questionnaire and exit cards are described in chapter 7.

Appendix includes tables and illustrations that were too large to fit in the main text. If the reader has trouble finding a correct illustration or table, it may be in the Appendix. There is also a brief description of the files accompanying this thesis.

The reader is expected to know the terminology covered by the Master's program of Computer Science, High-Performance Computing specialty, in addition to the basic terminology of computer graphics and web development.

2. The Course

The CGLearn environment was constructed with a certain layout of a computer graphics course in mind. That course (MTAT.03.015) was conducted in the spring of 2015. Parts of the system were also developed and tested during the Computer Graphics Seminar (MTAT.03.305) course that took place in the Fall of 2014 and together with the Computer Graphics course in the Spring of 2015.

The previous Computer Graphics course was conducted in the fall of 2013 by Konstantin Tretyakov and Ilya Kuzovkin, without a clear indication of the next time it would run. That time the course received a very high average feedback score of 4.7 from the students. Course comprised of weekly lectures and practice sessions on different computer graphics topics. Homework tasks were in C++, with either Allegro library for 2D graphics or OpenGL for 3D graphics. Students were graded based on a score, computed from the solutions to weekly homework tasks, a course project and the exam.

Because of the uncertainty of this course's next conduction, some of the students, who passed last time (me, Margus Luik, Ats Kurvet) teamed up with a modeling expert (Timo Kallaste) and a couple of other people interested in computer graphics (Jaanus Jaggo, Benson Muite) to create a good collection of materials and conduct the course.

Generally there has been a debate among educators, about how to teach computer graphics. One distinction is between a top-down and a bottom-up approach. Different educators have provided pros and cons for both ways [10]. The top-down approach usually starts, by introducing students to higher level software and libraries, moving down to specific algorithms and their implementation in the process. The bottom-up starts by first introducing the standard graphics pipeline and low-level algorithms, moving up to higher level software that uses them. Major arguments pro top-down approaches include the goal of the students to achieve an accountable result (*eg* a computer game) and to see, where and understand, how graphics are used in modern applications. In contrast, an argument against the top-down approach would be that students will lack the understanding of the basic low-level algorithms, because learning them may seem tedious after having already learned a lot of high-level techniques [11].

In this course this problem is tackled by introducing a mixed approach. The benefits of a top-down or bottom-up approach are largely student-dependent. That is why this course will start in a bottom-up way, but mid-semester students have a choice to either continue on the bottom-up path, or, alternatively, continue the course by learning higher level techniques in the modeling software Blender and the game engine Unreal Engine 4. This could very broadly still be classified as a bottom-up approach, but in the traditional manner, the high level software would be covered (if at all) only in a couple of last weeks. This was also the case with this course in the Fall of 2013. With the changed structure, students have the entire 8 weeks to get to know that software, while still other students (or even possibly the same ones, if they are interested) can continue on the bottom-up path and keep learning the different low-level graphics algorithms.

This is achieved by dividing the course to three modules:

- **Basic I** – Begins with low level algorithms and focuses on geometry, transformations, lighting, texturing and blending.
- **Basic II** – Continues with a variety of different graphics algorithms.
- **Game Engines** – Introduces students to Blender and Unreal Engine 4 and teaches the basic high-level techniques in them.

Each module lasts approximately eight weeks. Semester starts with the Basic I module and mid-semester students can choose to continue with Basic II, Game Engines or with both of them (Illustration 1). Continuing with both of the latter modules is an option for students, who are willing to spend more time to learn about the material from different perspectives simultaneously.

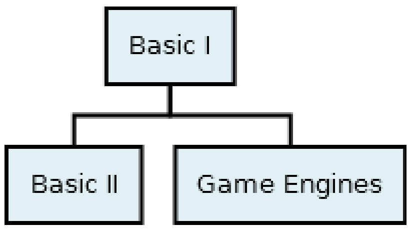


Illustration 1: Hierarchy of the three modules.

The course material is kept in two places. The CGLearn system holds the material with interactive examples, tasks with their scores, feedback and statistics, overall result progress, flashcards to help the students learn the material. The course page located at the Institute of Computer Science's Courses domain [12], holds the organizational and contact information, lecture slides, course schedule, project pages, external links. There is also a mailing list for general communication between the participants of the course and the educators.

In the rest of this chapter, thesis continues with an overview of the lectures, practice sessions and tasks, and the grading in the course. This is to explain the didactic approaches in those aspects of the course, and how the course was organized to support the modular layout.

2.1. Lectures

The lectures in the course cover mainly the basic ideas and concepts in computer graphics. During the first seven weeks, the lectures follow the topics of the Basic I module, covering it entirely. On the eighth week, there is a recapitulation lecture to go over the important aspects covered so far; help the students, who might have fallen behind on some specific topics; organize the projects. On the ninth and tenth week there are higher level lectures on the topics of Modeling and Game Engines, and Data Visualization. After that, until the fourteenth week, lectures will follow the topics of Basic II. That should be useful also for the students, who have chosen the Game Engines module. This is because several ideas in computer graphics need to be understood and applied also in a higher level software. The fifteenth week is for additional topics that the students can request and the conclusion of the course. The last week's lecture is dedicated entirely for the project presentations by students themselves. Refer to Table 15 in the Appendix for a complete list of the topics.

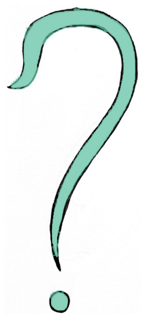


Illustration 2: Question mark that appears on the lecture slides.

Generally the lecture slides are constructed and lectures conducted in a supportive manner. This means that, although the key concepts are introduced and often derived, the main goal of the lectures is to look at the material in CGLearn from another perspective, discuss it together in a more student-oriented approach.

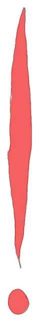


Illustration 3: Exclamation mark that appears on the lecture slides.

To favor this kind of discussion and alternative thinking among students, the lecture slides often contain questions or puzzles that students are asked to try and answer. These are indicated on the slides with greenish question marks (Illustration 2). They also serve as key points for the students, who have not attended the specific lecture and are looking at the slides afterwards. This also helps the students to recapitulate important thoughts from the lectures.

Another mark that appears on some of the slides is a red exclamation mark (Illustration 3). That indicates the most important concepts that the students should know about. Often times the amount of new information can be too much for the students to prioritize and organize. These marks help to indicate the ideas that should be learned first and foremost. This does not mean that other concepts are not important, but just indicates the concepts that give the most benefit in order to understand further topics.

At the end of each regular lecture, there are two questions asked from the students:

- What did you learn today?
- What more would you like to know?

Students are asked to write the answers to those questions on a small sheet of paper in approximately 5 to 10 minutes. This technique is described by Karm in [13], where it is called a *door pass* or *exit card*. These cards serve multiple purposes. First, the students themselves can look back at the lecture, and summarize the covered material. Secondly, the educator will get feedback on the quality of the lecture. Thirdly, it gives an opportunity for the students to ask questions anonymously. During the lectures, it turned out that many students wrote questions on the card, but were reluctant to ask them directly during the lecture. The questions were addressed and further explanation given to the students via a mailing list inside the ongoing week.

2.2. Practice Sessions and Tasks

There are two practice sessions occurring each week. During the first seven weeks, one of them is the JavaScript practice session, and another is the C++ practice session. The first one was conducted by me and the second one by Margus Luik. On the eighth week, both practice sessions cover an introduction to modeling software and were be conducted by Timo Kallaste.

After that, the first practice session is for the Basic II module and the second one is for the Game Engines module. In the Basic II module, me and Margus Luik conduct different topics and the choice of a programming language depends on the students preference. At that point students should have enough experience in their chosen language, to be able to grasp the ideas taught in the session and apply them themselves. The Game Engine module practice sessions were conducted by Ats Kurvet and Timo Kallaste, depending on the specific topic there.

The main goal of the practice sessions is to help the students understand the tasks in CGLearn and show them the initial steps towards progress in those tasks. This is didactically very important that the students should be able to see some progress made and estimate the effort required to make it. It is also important that the key ideas behind the techniques described in the tasks are universally understood among the students. Of course, there are other channels (*eg* the course mailing list) for the students to ask help about the tasks, but the practice session is considered to be the prime source for that.

Solutions to tasks can be uploaded to CGLearn and corresponding instructors can grade and send feedback for them. If a student has made mistakes in the solution, they have a possibility to resubmit a solution and thus maximize their points. There are two deadlines for the tasks. Basic I has a deadline mid-semester for all of its tasks. Basic II and Game Engines modules have a deadline in the end of the semester, one week before the exam. This gives students a freedom to plan their own time. Students are different and some are more self-learners, who prefer to do all of the tasks in one go. Others benefit more from a constant activity with the material and need to keep themselves on track each week. This freedom of deadlines should alleviate the problem of some students having a busy week with their other studies and thus not having enough time to complete the tasks in one week.

2.3. Projects

During the course, students are required to complete a project. There are no strict requirements for the actual nature of the project, rather than it be related to computer graphics. This allows students to pick their favorite topics and fix the scope according to their skills. Doing a project should help students consolidate the material learned and practiced. It also tests their ability to synthesize a solution for a given computer graphics problem. This is the highest level cognitive domain in Bloom's revised taxonomy [14]. It should also be quite rewarding for the students to create and see the solution to their own problem.

2.4. Grading

Grading for the course is done via a scoring system that distributes 100 points among the different gradable aspects of this course. There are also some opportunities to earn bonus points and achieve the overall score over 100.

During the course, the majority of the points accumulate from the tasks in different modules. Each module gives in total 20 points, and additionally some number of bonus points for extra tasks. Normally the student should be able to earn 40 points from the Basic I module and from either the Basic II or Game Engines module. If a student chooses to continue with both Basic II and Game Engines modules simultaneously, then a total of 60 points can be earned from the tasks.

The course project contributes 30 points to the score. These 30 points are given to every student from the start, but they will lose points if they miss important deadlines for the project. This approach was also taken in the last time with this course and was found to be effective.

The final exam contributes another 30 points and tests the general knowledge and understanding of the covered material. Participation in the exam is not mandatory, if a student has already earned a sufficient amount of points from other activities, then the student is considered to have worked enough for the grade that the current score would provide.

3. The CGLearn Learning Environment

This chapter first describes the different requirements that were considered, while analyzing the need for a custom environment. Based on the requirements, a couple of other learning environments were compared and analyzed. The same requirements logically guided the development of the system. After this, the implementation is described in separate sections for back end and front end. This chapter also includes the implementation description for the interactive examples. Those are also considered to be a part of the learning material in the next chapter.

3.1. Requirements

This section describes the primary requirements that were considered when comparing different existing systems. These requirements also served as main goals for the development. Although additional features and functionality was added, the main requirements were always the same. That additional functionality could also be formulated as requirements, but it seems clearer, to describe them in the Functionality section instead.

3.1.1. Functional

These are the main functional requirements that were considered in the start of this project.

3.1.1.1 Authentication

Student authentication had to be as easy as possible. It would have been a chore for the students to create another user account specifically for this system. It would have also been insecure and even more tedious to allow students to submit solutions and specify their student number of other indicator in the process.

This meant that authentication had to be based on the university's credentials.

3.1.1.2 Material

The system had to allow easy writing and reading of the computer graphics related materials. The material needed to be grouped and categorized in some logical hierarchy. Because the course was planned to have a mixed bottom-up and top-down approach that included different modules, then this kind of modular structure of topics and their corresponding materials was a clear goal.

3.1.1.3 Interactive Examples

Successful teaching should include interactive (engagement level 4, see [15]) examples for students of most of the material and concepts covered. This meant that each material should have several interactive computer graphics examples accompanying it. System had to support the creation of such custom examples and provide an easy inclusion of them inside the material.

3.1.2. Non–Functional

3.1.2.1 Platform

After the development started, the platform of choice was Ubuntu 14.04 with Apache HTTP Web Server 2.4.9, PHP 5.5, MySQL 5.5. This was the setup in the server, where the project is hosted on, and suits the project's requirements nicely.

3.1.2.2 Browser Support

The entire system had to work with the latest version of Chrome browser [16] and the version that was in the lab computers in the institute. The examples should be runnable on the university laptops that have at least the integrated Intel HD 4000 graphics adapter.

3.1.2.3 Response Time

All functionality in the system should have a sufficiently low response time. No strict limit for the requests was made, but the estimate is that no student request should take more than 2 seconds (with the exception of a file upload). To account for this requirement in CGLearn, most of the database queries were monitored and indexed. Additional steps can be taken to ensure the responsiveness, these include enabling PHP-s APC [17] cache and using the Require.js optimizer [18] to minify and combine JavaScript.

3.2. Existing Solutions

The need for a custom system arose from the main requirements to be able to construct a modular course and implement specific interactive JavaScript examples for the material. System, that would support them, should also support the basic requirements for a standard course. These would include student authentication, the functionality to submit solutions for tasks, student and teacher being able to exchange feedback for the solutions, a grading system that is easy to use for the teacher, and visible and transparent for the students.

Couple of choices for already existing learning environments were considered, when analyzing the need for custom development. Those included the use of a *Courses* page, the learning management systems Moodle and Udutu.

3.2.1. Courses page

The Institute of Computer science has a *Courses* domain, where most of the courses have a page that serve as the communication point between the students and the teachers. Teachers have the ability to modify the course pages, which are based on PmWiki [19]. Usually this ability is utilized to provide lecture and practice session materials for the students, convey the general descriptive and organizational information of the course, provide additional links and references. Those pages allow free access for course materials to anyone from outside the university, which is certainly a plus.

Often times, the students are also able to submit homework solutions via the page and additionally can have their own sub-pages, which they have to modify (write lecture notes, project descriptions etc).

Submitting the homework solutions using the current system has three major problems. First, it consists only of a file upload and a comment field. When a student uploads a new solution, the previous one, together with the previous comment, will get overwritten. This creates a situation, where asking the student to correct their insufficient solution, while preserving the history of the previous submission, is impossible. Secondly, the system also does not have a way to actually grade the solutions. Usually the grades are entered to a separate spreadsheet by the teachers manually, which creates an overhead and is prone to human error. The third problem is that, because this is just a file upload, with no specific database built around it, asking students for additional information about the solution (eg, how much time it took, how difficult it was), would consist of students writing the answers as text. That creates a lot of unnecessary work for the teacher to do statistics and estimate the current tempo and difficulty of the course.

Although creating interactive custom JavaScript examples, would have been possible in that system (as it is done in MathWiki [20]), it would have probably taken the same amount of time, as implementing the entire system from the scratch, with a specific purpose.

Another problem would have been with material creation. The PmWiki markup language is too limiting for applying a consistent style of some accepted front-end framework (such as Bootstrap).

A Courses page is still used in the Computer Graphics course for providing the organizational and descriptive information. This offers a quick access to the schedule for the students registered in the course, and an overview for all other students. Lecture slides are added to the course page, because most of the students are used to that, and it is effective to provide students a way of exchanging material they are familiar with.

We also use the course page for the sub-pages of student projects. Each team gets their own page that they have to write a description of the project to. This serves as a good public page to advertise both their project, and the course itself.

3.2.2. Moodle

University of Tartu uses the learning platform Moodle [7] to conduct some of the courses. Moodle provides a very extensive set of didactic tools to facilitate learning, and thus should be considered when a specific layout of the course is in mind.

Regarding the main requirements of this course, the modular layout of the materials would have been certainly possible in Moodle. Unfortunately creating custom JavaScript examples would have not been possible without extensive communication with the support and specific feature requests. Moodle only supports material with engagement level 3 (responding), while our requirement was engagement level 4 (changing).

There were also additional problems with the use of Moodle that arose from secondary requirements. Unfortunately, the solutions submitted for tasks in Moodle, can not be grouped by the teacher in a way that filters the solutions that need a response (see Table 1). Those solutions would be the ones that are submitted and ungraded. Moodle allows to filter solutions that are not submitted (does not matter, if graded or ungraded), and also the solutions that are ungraded (does not matter, if a solution is submitted or not). This creates a lot of overhead for the teacher to manually filter out the solutions that need to be graded among the ones that are not submitted.

Table 1: Possible filters in Moodle. It is not possible to filter only the submitted and ungraded submissions. It is possible to filter with A the ungraded submissions, and with B the submitted solution, but not the intersection.

	Graded	**Ungraded**
Not submitted	*Not applicable*	A
Submitted	B	A, B

Another problem was that the Flashcards didactic tool had an insufficient design and problematic functionality for the teacher. The design included instant switching of flashcards and turning them over. This kind of switching, without a corresponding animation, can confuse the students about what is actually going on. Furthermore, the design included a gray background for the card, which meant that the contents with a black font were not brought out in an easily readable way. Functionality problems included the generation of 40 dummy flashcards upon the initial creation, which the teacher then had to either fill or manually delete one-by-one. Although, this problem was not solved in CGLearn, the overhead caused from it in Moodle, did not outweigh the decrepit view of the flashcard to the students.

3.2.3. Udutu

Another learning management system that is advertised to allow conditional branching of the learning material is called Udutu [21]. While experimenting with the different features that Udutu provides (especially the TreeView course map), I managed to create a situation, where publishing the material resulted in a system error.

Also, there is no documentation on how to add custom JavaScript to the materials. The Frequently Asked Questions section [22] does mention that there is an API and an ability to create custom JavaScript, but I could not find publicly available documentation for it.

Generally, Udutu does provide a lot of different possibilities for engagement level 3 (responding) interactions. The responses to such engagements can direct students to different parts of the material, depending on the correctness of the response. Although that would be beneficial for the learning process, it does not fit with the requirements.

3.2.4. Conclusion

The examined existing solutions did provide many different tools that would generally enhance the learning process and give courses more value. The specific learning management systems (Moodle and Udutu) provide educators with the ability to create engagement level 3 (responding) interactive material. Although certainly better then just engagement level 1 (no viewing) or engagement level 2 (viewing), they do fall short for my requirement of being able to modify parameters and see different effects.

The Courses page would have allowed the creation of content with a higher engagement level, but it would have created a lot of overhead for simpler things, like exchanging feedback with students or doing result statistics. Because of the PmWiki formatting language, creation of simple content is easy, but more complex formatting would include a lot of overhead, compared to actual HTML and CSS.

Both the Courses page and Moodle would have satisfied the authentication requirement.

Because of different problems with each of the examined learning environments, a custom environment called CGLearn was built from scratch.

3.3. Implementation

CGLearn was implemented in PHP-s Laravel Framework 4.2 [23], using a MySQL database and Doctrine 2 ORM [24]. Authentication is done via Shibboleth Service Provide Apache module [25]. Front end is based on Require.js [26] for modular script loading. It uses jQuery 2.1 [27] and Bootstrap 3.4 [28]. Interactive examples are done with Three.js r70 [29] and formulas are shown with the latest version of MathJax [30]. Statistics charts use HighCharts [31] and modal popups are created with FancyBox 2.1.5 [32]. Specific benefits and design choices are discussed in more detail in the corresponding subchapters.

As with any web environment, it consists of a back end and a front end. The specific structure of both parts is described in the following sections. This description should give an overview of the architecture decisions and also serves as partial documentation for the system.

3.3.1. Back End

Server side of the application was implemented in PHP using the Laravel Framework version 4.2. Although currently version 5.0 has been released, 4.2 was the latest release at the start of the development. Choice of the Laravel Framework over Zend Framework was made because Laravel is considered to be suitable for non-enterprise level web development. It also follows the convention over configuration principle, which means that less time can be spent on writing the different configurations, compared to Zend that uses the configuration over convention principle [33].

With that in mind, Laravel 4 by default is lacking in some areas that would benefit the current project. One of those is that out of the box it does not support modular development without specific additional effort. An extension [34] for Laravel 4 modules was added to the project to get that support.

A second concern was with the Eloquent ORM that is the official ORM for Laravel. Eloquent follows the active record pattern, which means that the domain objects and database queries are bundled together in single classes. This usually comes down to personal preference, but for me it seems more logical to use the data mapper pattern, which has the domain objects mapped to the database, and separate repository classes handle the queries for those objects. For this reason the Doctrine ORM was added to the project and a specific package that allows Laravel to communicate with Doctrine [35] was included.

Application is hosted on a Linode [36] instance that runs Ubuntu 14.04 [37] and the Apache HTTP Web Server 2.4.9 [38]. The instance belongs to a web-development company Blue Lynx OÜ [39]. Reason for using their server lies in a better responsiveness and familiarity of that company compared to similar services provided by the university. The entire project also uses a private Git repository, hosted in the same server.

3.3.1.1 Application

Application is divided into two modules and a general part that is common for both of them. That general part consists of the domain objects (called entities) and their corresponding repositories. There is also general user-specific functionality in the form of a few controllers and services. Event listeners that deal with sending e-mails, are similarly general for both modules.

There are two use cases that occur for both students and teachers. The first one is the initial landing page, which has the request redirected to the students landing page. This is because generally teachers are also interested in ending up in the material sections visible to the students, instead of the teacher's administration panel. The second use case is logging out that, upon success, shows a same page for both of the user groups.

With that in mind, the teacher's group is more of an extension of the student's group. Teacher users have most of the same functionality, to view and navigate the material. They do, however, have restrictions to submit solutions. Compared to students, teachers have access to the administration panel that includes additional functionality related to course management and educating.

Detailed descriptions of the student and teacher modules will illustrate more of the different functionality those user groups have in CGLearn.

3.3.1.1.1 Student Module

This module has functionality related to displaying the course material, creating task submissions and feedback, showing results. This functionality is available for all users of the system, who have authenticated themselves via Shibboleth. Although a couple of specific functions, such as submitting solutions for tasks, are restricted to students only.

Controllers

Controllers for parsing requests from routes commonly prefixed with *student/* prefix. Their actions in this module are explained in an alphabetical order.

FlashcardController

Is responsible for parsing the requests related to flashcards.

Actions:

- *index($slug)* – Parses the request to fetch the user-specific deck of flashcards for a given module (specified by the slug). Returns a view with all the flashcards ready for display for the current user.
- *answer()* – Parses an Ajax request that needs to include *id* of the answered flashcard and *score* that specifies the student's response to it. Returns a JSON response that has a new number of flashcards for this module. This is used to update the number of flashcards in the left menu.

HomeController

Has actions that deal with functionality available on the main landing page of students. Also includes profile viewing and editing actions.

- *index()* – Parses the request for the landing page of students. Returns a view containing the current course information for the current student.
- *profile()* – Parses the request for the profile view. Returns the current student's profile view. Could be extended for showing the profiles of other students, but this has not been a main concern of the system.
- *profileEdit()* – Allows the current student to edit the profile. Parses the requests for showing the editing form and saving new profile data, which is also validated. For GET requests the editing form is shown. In the case of a successful edit, the response is redirected to the profile route.
- *logout()* – Parses the logout request: clears the session and redirect to Shibboleth's logout URL.
- *changeCourse()* – Parses the request to change a course. On a successful change, the chosen course is saved to be the active course for the current student and a student is redirected to the landing page.
- *changeCourseModule()* – Parses an Ajax request that specifies a chosen course

module to be the active course module for the current student. Returns a JSON with a success flag.

MaterialController

Has actions that deal with showing of different parts of the material.

- *index()* – Parses the request to show the main view of the material. The current course is used to show its description and the modules in it. The left material menu is constructed and a current course module is used to open a specific module in it.
- *topic($slug)* – Parses the request to show a specific topic inside the material. The slug parameter of a topic is used to fetch the topic. Response is a view without the layout, because topics are loaded via Ajax.

ResultController

Has actions that deal with the results table viewing.

- *index()* – Parses the request to show the results table. Also calls functions to populate and update the results table. Results are fetched and the response is modified so that only the current user has a correct name in the results table. Other students will get a fake name.

StatsController

Has actions that deal with the statistics plots viewing.

- *index()* – Parses the request to show the statistics page. Calls functions to fetch the corresponding statistics and also to update the statistics.

TaskController

Has actions that deal with the viewing and submitting of tasks or their feedback.

- *index()* – Parses the request to show the tasks tree. Current course is used to fetch the corresponding modules and a tasks tree is generated based on the ordering of modules, topics, materials and task dependencies.
- *details($slug)* – Parses the request to show the details for one task, specified by the

slug. Also shows the data about different submissions to this task, by the current student. Includes a submission form is the current user is a student.

- *addSubmission($slug)* – Parses the request to add a submission to the task specified by the slug. Submission data is validated and the *submission.create* event is fired. Response is redirected to the task details route.
- *downloadSubmission($id)* – Parses the request to download a submission made by the current user and specified by the id. Current user is validated and a download response, that has the contents of the submitted file, is returned.
- *commentSubmission($id)* – Parses the request to add a comment to the submission of a current user and specified by the id. Comment and the current user are validated and on success, the response is redirected to the task details route.

Services

Services encapsulate the business logic used by the students' actions. Services and their functionality is explained in an alphabetical order, with the respect to namespaces. Most services have a *get($id)* and *getBySlug($slug)* methods that are not explicitly mentioned here.

Many functions also have the user as an optional parameter. If nothing is assigned there, then usually the current user is fetched as the user. Also, here only the public functions are described. Many services also have protected functions that are being used by the public functions.

Some of the functionality described here, may be actually performed by the repositories that the services call.

CourseService

Deals with business logic related to courses.

- *getCurrentCourse(AbstractUser $user = null)* – Gets the current course for either the specified user or the current user, if none is specified. First the session is probed to see, if there is a current course id specified there, if there is that course is returned. Secondly user preferences are probed to get a current course from the database, if it is there, then that course is returned. Finally, is none of those methods found a current course, a default course is returned from this service.

- *getDefaultCourse()* – Finds the first course from the database that has the *isHidden* flag as false.
- *setCurrentCourse(AbstractUser $user = null, Course $course)* – Sets the current course to the session and the database of the current user.
- *getActiveCourses()* – Returns an ArrayCollection of all courses that have the *isHidden* flag as false.
- *getCourseGradeFromScore($score)* – Converts a score into a grade letter. If the score is above 100, the letter „A“ is returned. If the score is below 50, the letter „F“ is returned. Otherwise the score is divided by 10 and a value from an array containing the letters from „A“ to „F“ is fetched by taking the key $10 - ceil(\$score)$.

Course/ResultService

Has the logic to display the course results to students. This service extends the AbstractResultService in the general part of the application.

- *getCourseStudentResultsByCourseForStudent(Course $course, AbstractUser $student)* – Fetches the results from the parent service and creates a Faker. Next it goes through the results, detaches them from the ORM, and whenever the results are for a different student, replaces the name by a fake name generated by the Faker. Then returns the modified results.

FlashcardService

Has the logic for the flashcards.

- *initialize(Module $module, AbstractUser $user = null)* – Checks if the user has the flashcards generated for the specified module. If they are not, then generates the user flashcards, based on the flashcards assigned to this module. The current date with the time specified as 00:00 is assigned as the *nextDate* for each user flashcard.
- *getForDisplay(Module $module, AbstractUser $user = null)* – Gets the flashcards

for the user that have the *nextDate* less than the current time.

- *countForDisplay(Module $module, AbstractUser $user = null)* – Counts the flashcards for the user that have the *nextDate* less than the current time.
- *saveAnswer(UserFlashcard $userFlashcard, $score)* – Takes into account the score, given by the user for this flashcard, to calculate the new *nextDate* based on the SuperMemo 2 algorithm [9].

ModuleService

Has different getters for modules.

- *getRootModules(Course $course)* – Gets the root modules in the module tree for the specified course.

StatsService

Has the logic for constructing the statistics.

- *getTasksWithCurrentSubmissions()* – Gets the tasks that a current user has submitted a solution to. That submission is assigned to be that task's only submission.
- *getTasksWithExtremeScores()* – Gets tasks, and assigns the submissions with a minimum and maximum score to be their only submissions.
- *getTasksWithExtremeDifficulties()* – Gets tasks, and assigns the submissions with a minimum and maximum difficulties to be their only submissions.
- *getTasksWithExtremeTimes()* – Gets the tasks, and assigns the submissions with a minimum and maximum time estimates to be their only submissions.

TaskService

Has the logic that deals with tasks.

- *getTasksByCourse(Course $course = null)* – Gets all the tasks for the course (or the current course, if none is specified).
- *updateAverages(ArrayCollection $tasks)* – Goes through the ArrayCollection and

checks, which tasks have the flag *averagesNeedUpdate* set. For those tasks, the average values for the score, difficulty and time estimate are calculated and updated.

- *getMainTasksInTopic(Topic $topic)* – Gets the tasks in a topic that have no prerequisite tasks.
- *getSuccessorsInSameMaterial(Task $task)* – Gets the tasks that have the given task as a prerequisite, and also are assigned to the same material.
- *hasPrerequisitesMet(Student $student = null, Task $task)* – Checks, if the student has the prerequisites met in order to submit a solution for the given task. If there are preceeding tasks that are set as a prerequisite for this task, and a student has not submitted a solution for them, this function returns false.

Task/SubmissionService

Has the logic that handles task submissions.

- *getLatestSubmission(Student $student = null, Task $task)* – Finds the latest submission of the student to the task.
- *save(Task\Submission $submission = null, $data)* – Saves a new task submission. Sets the corresponding values from the data array, sets the *averagesNeedUpdate* flag in the task, sets the uploadable file info to the entity. Also checks if there have been previous submissions to the same task by the same student. If so, those are updated to no longer be the latest.
- *getTotalPointsInModule(Module $module)* – Gets the total number of points, the current student has received in the specified module.

Task/Submission/FeedbackService

Has the logic to handle task submission feedback.

- *save(Feedback $feedback = null, $data)* – Saves the feedback information.

TopicService

Has no specific logic, besides just a getter for a topic by slug.

3.3.1.1.2 Teacher Module

This module has the functionality to modify the course materials, grade task submissions and create feedback, modify course results and students. Some limited actions do not have a programmatic functionality in the system. For example, the creation of teachers is done by hand. Those features could be further developed to become a part of this module, but currently, they were not that essential.

Controllers

Those controllers parse the requests commonly originating with a *teacher/* prefixed routes. Such routes have a further authentication filter (besides Shibboleth) assigned to them, to allow access only to a limited number of users – those that are teachers.

Most controllers have an *index()*, *edit($slug)* and *update($slug)* actions. Those are for displaying a list of the entities, displaying the editing form and saving the updated data. Controllers that have only that functionality are: ***CourseController***, ***FlashcardController***, ***MaterialController***, ***ModuleController***, ***TopicController***. These follow a common structure and are not individually described here. Other controllers are described in an alphabetical order.

Course/ResultController

Has the actions for showing and editing the results table.

- *index($courseSlug)* – Parses the request to show the results table, for the course specified by the slug. Also calls populate and update functions on it.
- *modulePartial($courseSlug, $moduleSlug)* – Shows a results table for one module, specified by the slug, in a course, specified by the slug. This action returns a view that is in a popup layout, because it is shown in a Fancybox modal.
- *update($courseSlug, $resultId)* – Parses the Ajax request to update the score in one of the results. The new score is saved, and a new total score is also calculated. Action returns a JSON response that has the updated data for the row in the results table.

Course/StudentController

Has the actions for assigning specific students registered in the system, with a specific course. This is needed to generate the results for the students for a course.

- *index($courseSlug)* – Parses the request to show all the students currently assigned to the course specified by the slug.
- *import($courseSlug)* – Parses the request to assign students to the course, based on a CSV exported from the Study Information System (SIS) [40]. The CSV data needs to be in the payload of the request. Response redirects to the *import.result* route.
- *importResult($courseSlug)* – Parses the request to show the results of a CSV import. Response has a view that shows the number of found students, not found students and added students.

Task/SubmissionController

Has the actions for listing the submissions, grading them, exchanging feedback, downloading solutions.

- *indexNew()* – Parses the request to list the submissions that need a response from the teacher. Response view has that list.
- *details($id)* – Parses the request to see the detail view of the submission specified by the id. Also marks the feedback assigned to this submission as read by the teacher. Response includes a view that has the detail view of that submission.
- *download($id)* – Parses the request to download a solution specified by the id. Response returned is a download, with the contents of the solution file.
- *comment($id)* – Parses the request to add a new comment to the submission. Fires the *feedback.create* event. Response redirects to the details route.
- *bulkDownloadNew($topicId = null)* – Parses the request to download all the submissions that need a response from the teacher. If a topic id is specified, it filters only the submissions in the corresponding topic. This action fetches all the corresponding submissions from the database, then finds their files in the file

system. Those files will be unpacked (if they are packed with ZIP or RAR). After this a file tree structure is created that lists the tasks in the first level, students in the second level. The solution files are moved to the corresponding folders and the entire tree is then archived again to a *bulk.zip* file. The contents of that file are downloaded in the response.

- *grade($id)* – Parses the request to assign a grade to a submission specified by the id. On success, the *submission.grade* event is fired. Response redirects to the submission details route.
- *gradeAndComment($id)* – Parses the request to assign a grade and create a feedback in the same time. This request comes from the *Quick Grade* functionality in the new submissions list. On success, the *submission.gradeAndComment* event is fired. Response JSON has the *success* flag, new score and the id of the submission indicated. Based on those, the row in the new submissions list is later updated.

StudentController

Has only the *index()* action that displays the list of all student users in the system.

TaskController

In addition to the actions described above, it has also actions for adding new tasks and deleting tasks.

- *add()* – Parses the request to add a new task. Request must have a material specified in the payload. Returns a response with a form that asks information about the task.
- *insert()* – Parses the request to add a new task. Request must have a payload that specifies the details of that task. That payload is validated and on success, the task is created. Response then redirects to the task index.
- *delete($slug)* – Parses the request to remove a task, specified by the slug. This action returns a confirmation view.
- *remove($slug)* – Parses the request to remove a task, specified by the slug. This action removes the task.

TeacherController

This controller has actions that start or end specific modes available for the teacher user.

- *startGhost($id = null)* – Starts a ghost mode for the user, specified by the id. If no id is specified, an arbitrary id of a dummy student is picked. Ghost mode allows the teacher to see the student view from the perspective of a specific student.
- *endGhost()* – Ends the ghost mode.
- *startDebug()* – Starts the debug mode. That mode has a PHP debugbar [41] available and debugging enabled for each subsequent request.
- *endDebug()* – Ends the debug mode.

Services

Services here specify the business logic only available for the teachers. Most services again, have the common get($id), getBySlug($id), *getAll()* functions. Also here there is a *save(Entity $entity, $data)* function that occurs often. This function allows to edit the data of a specified entity and saves it to the database.

Services, with only that functionality are: ***CourseService***, ***Course/ResultService***, ***FlashcardService***, ***MaterialService***, ***ModuleService***, ***TopicService***. Services, with additional functionality are described in more detail.

Course/StudentService

Allows to get and add students for a course.

- *getAllByCourse(Course $course)* – Finds all of the students, who are assigned to the given course.
- *getByCourseAndStudent(Course $course, Student $student)* – Finds the Course/Student entity for the specified course and student.
- *addMany(Course $course, $students)* – Assigns the given students to a course, if they are not already assigned. Returns, how many of the given students were assigned.

Task/SubmissionService

Has additional functions to get the submissions.

- *getAllLatest(Task $task = null, Topic $topic = null)* – Gets the latest submissions, given a task and a topic. If the task or a topic is missing, filtering is not performed.
- *getAllLatestNew(Task $task = null, Topic $topic = null)* – Gets the latest submissions that need a response from the teacher. If a task and a topic are given, the results are filtered.
- *getLatestSubmissionsInModule(Student $studnet, Module $module)* – Gets the latest submissions for a student for all tasks inside a specified module.

Task/Submission/FeedbackService

Has an additional function to mark the feedback read.

- *readFeedback($submissions)* – Marks the feedback read by the teacher for all the submissions given.

StudentService

Has an additional function to find students based on the data from a CSV file exported from SIS.

- *getFromImport($csv)* – Tries to find a student based on the data in the CSV, where each row conveys the data for the student in the format outputted from SIS. First it tries to find a student by the first and last names. If there are multiple occurrences of the students with a same first and last name, then this fails. If the name look-up has failed, then it tries to find a student based on one of the two e-mails outputted from SIS. If this also fails, then there is currently no way to locate the student from CGLearn. Function outputs an array, with keys *found* and *notFound*. Found students include all the entities of students that were found. Not found array includes all the rows of the CSV that failed.

TaskService

This service has additional functions for finding specific tasks and deleting tasks.

- *getAllSortedInMaterials()* – Gets the tasks in the alphabetical order of the topic's, material's and task's titles.
- *getTasksByCourseAndModule(Course $course, Module $module)* – Gets the tasks beloning to the specified course and the module.
- *getTasksByTopic(Topic $topic)* – Gets the tasks in the corresponding topic.
- *delete(Task $task)* – Tries to delete a task and set the preceding task of the next tasks to the predecessor of the current task. When a task has submissions, this function will fail. Currently, tasks are hard-deleted. Further development could be done, to make them soft-deletable.

TeacherService

Has additional functions for getting specific teachers.

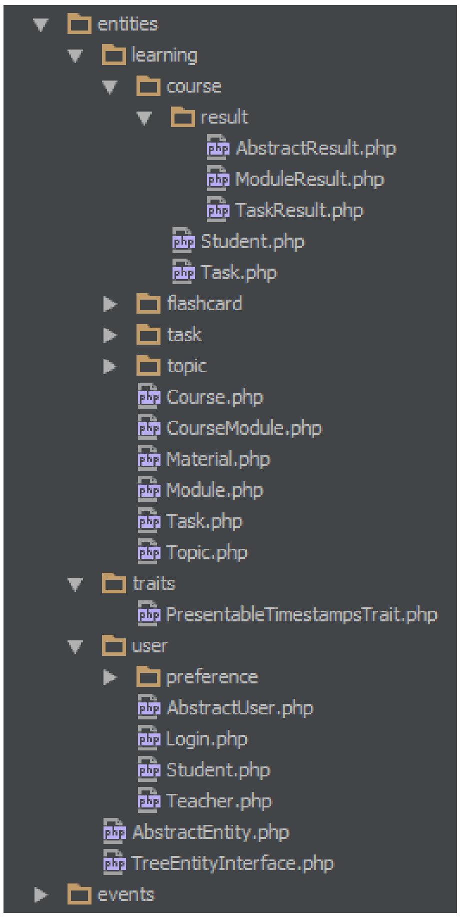
- *getTeachersForSubmission(Submission $submission)* – Gets the teachers that are responsible for the specified submission.
- *getTeachersForCurrentCourseModule(Module $module)* – Gets the teachers assigned to the specified module and the current course.
- *isAuthorized()* – Returns a boolean, specifing if the current user should be allowed to access the teacher module. This is used in the authentication filter.

3.3.1.2 Database and ORM

As mentioned before, because of the logical separation of entities and repositories in the data-mapper pattern, Doctrine 2 ORM was chosen instead of Eloquent ORM. Relational database MySQL 5.5 is used to store the data. MySQL Workbench 6.1 [42] was used to construct and modify the database model (Illustration 84 in Appendix), thus effectively also the domain model.

In addition to Doctrine's own annotations, an extension package called Doctrine Extensions [43] was used to include support for tree structures (Tree), slugs (Sluggable), file uploads (Uploadable), sorting (Sortable) and timestamps (Timestampable).

The domain model consists of Doctrine entities, which are generally constructed and mapped according to the database model. There are some aspects that can not be easily read from the database model. Those specifics are described in the following sections.



```
▼ entities
    ▼ learning
        ▼ course
            ▼ result
                AbstractResult.php
                ModuleResult.php
                TaskResult.php
              Student.php
              Task.php
        ▶ flashcard
        ▶ task
        ▶ topic
          Course.php
          CourseModule.php
          Material.php
          Module.php
          Task.php
          Topic.php
    ▼ traits
          PresentableTimestampsTrait.php
    ▼ user
        ▶ preference
          AbstractUser.php
          Login.php
          Student.php
          Teacher.php
      AbstractEntity.php
      TreeEntityInterface.php
▶ events
```

Illustration 4: File tree inside the entities folder.

3.3.1.2.1 AbstractUser, Teacher and Student

All users that log into the system via Shibboleth, are children of the *AbstractUser* class. By default they will become instances of the *Student* class. If they do not have a mapping to the Student class, but instead to the *Teacher* class, then they are considered to be teachers. This is achieved via the class-table inheritance [44] of Doctrine. The *users* table is joined to the *teacher* or *student* table, based on the specified *discr* value. This allows the data common to all users (their university username, e-mail, etc) and associations (preferences, logins) stored in one table and the corresponding role determined by the specific inheritance.

Similar pattern is used for the *AbstractResult*, *ModuleResult* and *TaskResult* classes.

3.3.1.2.2 PresentableTimestampsTrait

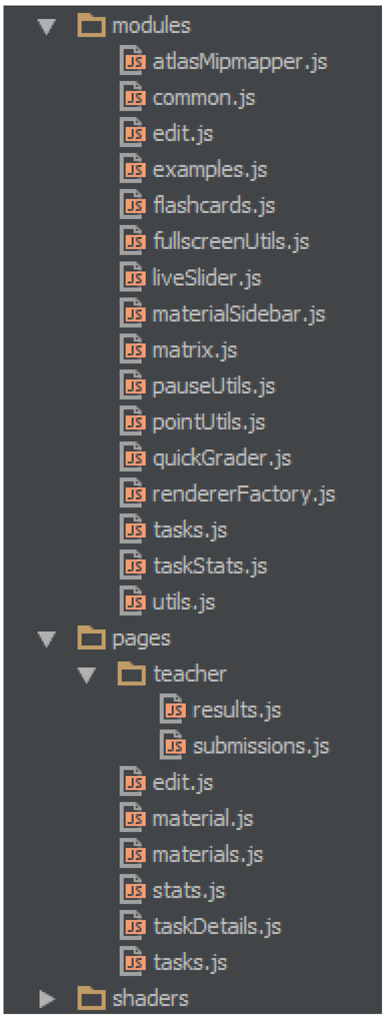
Because the datetime values are stored in the UTC timezone, there is a need for a conversion of the datetime values, when actually showing them to the users. With Laravel's own Eloquent ORM, this would be solved by using Carbon extension for the PHP-s DateTime class [45]. Instead of trying to integrate Carbon and Doctrine, there is a *PresentableTimestampsTrait* that is cloning the *DateTime* inside the entity, setting the correct timezone and returning a new *DateTime* object. This trait assumes that the entity has already the *Timestamps* trait that creates the *createdAt* property.

3.3.2. Front End

Design of the front end uses the Bootstrap framework. This causes the style to be generally uniform and pleasingly clean. For the sliders in the interactive examples, a slider plugin [46] was added. That plugin was extended with the *liveSlider* class, to allow the on slide update event to be triggered after a time interval. Otherwise, the slide event was triggered too often, when sliding the slider, and that caused slowness in performance.

Mathematical formulas and constructions were written in Latex and rendered with MathJax library. That library is included to fetch the newest version directly from their source. This has once caused a bug in one of their releases [47] to propagate to CGLearn. On the other hand, newer features, like the faster HTML+CSS renderer and other tweaks, have also been updated without additional effort. MathJax seems to be a stable library, when the use of Latex is considered. They can be trusted not to deviate from the standard Latex formatting in their newer releases.

All of the JavaScript is loaded dynamically with Require.js. Reason for its use is that, because there are pages loaded via Ajax that have different interactive examples in them, the JavaScript of the examples should also be dynamically loaded. It would be a waste of resources, if the page would load all of the examples in on the initial load, because most of the users do not need all that code during their session. Another way would be to manually configure the different pages to also load specific JavaScript. Require.js provides a clean and structured way to do just that.



```
▼ modules
    atlasMipmapper.js
    common.js
    edit.js
    examples.js
    flashcards.js
    fullscreenUtils.js
    liveSlider.js
    materialSidebar.js
    matrix.js
    pauseUtils.js
    pointUtils.js
    quickGrader.js
    rendererFactory.js
    tasks.js
    taskStats.js
    utils.js
▼ pages
  ▼ teacher
      results.js
      submissions.js
    edit.js
    material.js
    materials.js
    stats.js
    taskDetails.js
    tasks.js
▶ shaders
```

Illustration 5: File tree of the JavaScript files inside modules and pages folders.

Entire custom JavaScript is grouped under three categories: pages, modules and examples.

3.3.2.1 Pages and Modules

Specifying JavaScript modules for different pages is one of the practices of Require.js [48]. Each page, that needs some specific JavaScript, has only the corresponding page module required in the HTML. This way it does not matter, if the page is loaded via Ajax or not.

Within the corresponding page module, the corresponding functionality modules are required. For example, the left menu in the material page is handled with *MaterialSidebar* module. So the materials.js page module has in it the dependency of the *MaterialSidebar* module and creation of a new *MaterialSidebar* instance.

The modules specify a component or a small piece of functionality in the front end. That has its own dependencies and implements a specific functionality for that component. Pages specify the modules, that a specific HTML page has in it.

3.3.2.1.1 Common

Has in it currently only the listener for the form that changes the course in the upper left corner of the page. Whenever the select value in that form is changed, the surrounding form is submitted.

3.3.2.1.2 Edit

Initializes the TinyMCE [49] editing for textareas with a corresponding class. This component is used extensively in the teacher's area of CGLearn. All the material can be written and modified through TinyMCE.

3.3.2.1.3 Examples

Has dependencies for the *RendererFactory*, *FullScreenUtils* and *PauseUtils* modules. This is the module that starts and runs all of the interactive examples. Whenever the *init()* function of this module is called, it will search under the *#material-content* selector, for all the elements matching a *.example-box* selector. The *data-example-slug* attribute of the matched elements is then used to require the corresponding example from the *examples* folder.

For each of the examples, a renderer is asked from the renderer factory. This is because, there is a limited number of renderers (WebGL contexts), that can be created.

After constructing an instance of the example, a THREE.Scene is created and an array named *meshes* from the example is probed. If there is no such array, there should be an attribute *mesh* in the example. Those attributes need to contain the objects that will be added to the scene.

Example must also define an *addControls()* method that gets called to add the specific controls to the HTML element containing the example. At this point, the <canvas> element from the renderer is also added to the HTML and the *FullscreenUtils* and *PauseUtils* are called to have them add their own controls if the example allows them.

If the example provides a *camera* attribute and it holds an instance of THREE.Camera, then its value is used for the camera. Otherwise it may also hold a keyword *orthographic*, in which case a default orthographic camera is constructed. If none of those cases happened, then a default perspective camera is constructed.

Next, the animation event loop is made. In it there are checks to see if the example still needs time to load (*isReady* flag) or is paused (*isPaused* flag). Also, if the example is in full screen, then the *FullScreenUtils* are called to render it. Otherwise the usual *renderer.render(scene, camera)* is called.

Before reaching the end of the animation loop's iteration, the example is probed for a method *update*. If there is such a method, then it will be called after the initial rendering by the *Examples* module. This update method can be (and is) used to specify different animations and movement inside the example itself. At times it also happens, that the example needs to do more rendering passes. The *update* method is also provided the renderer, scene and the camera objects for such cases.

There is also a *clear* method in the *Examples* module for the case, when the page, that included some examples, will be unloaded. Because loading and unloading of the material is done via Ajax, then we need to stop the previous event loops and free the renderers.

3.3.2.1.4 Flashcards

This module provides the functionality for the flashcards. The *init()* method will find all elements matching *.flashcardWrapper* from the element that matched *#flashcardsWrapper*. For all of the flashcard elements, event listeners are bound for the turn button and score buttons. On pressing the turn button, the flashcard is rotated 180° around the y axis. On pressing the score button, the score request is sent via Ajax in the background, while the card is moved upwards and transparency increased. When that animation has ended, the next element, that matched the *.flashcardWrapper* selector is shown. If the last card was answered, then another specific element is shown, that indicates the end of the deck.

3.3.2.1.5 FullScreenUtils

The *FullScreenUtils* module adds a button above the examples, that allows the example to be viewed in full screen mode, if it is supported by the corresponding browser. When that button is pressed, the full screen mode for the example is set. The controls, that were specified by the example, are moved to the top left part of the screen and shown on top of the render.

In the full screen mode, there are some possibilities for most of the examples. The default is that the smaller render is done on a texture and that texture is then rendered full screen, with either the nearest neighbour or bilinear interpolation. Reason for this is that often times some of the examples need to preserve the relative size of their objects. For example, to show with a projector in the lectures.

Second option is to switch the actual rendering to have the dimensions of the entire screen. This causes 2 pixel thick lines to actually be 2 pixels thick. Depending on the example, it may improve the quality, or it may not, as is the case with the procedural noise example. In that example, the noise is sampled so that the result would look good on the smaller resolution. There are two examples, the ray tracing and path tracing, for which in the full screen mode, changing to the actual fulls screen rendering of the example, is disabled. This is because, those examples are already computation heavy and the complexity depends on the number of pixels. Examples can set an array *fullscreenModes*, to specify, which modes they want enabled or disabled for them.

3.3.2.1.6 LiveSlider

This is the module that extends the Bootstrap Slider plugin. It provides the ability to specify the *timedUpdate* key in the configuration, when initializing a slider. This timed update listener is called if the slider gets a *slide* or *slideStop* event. It checks, if an interval has passed from the last call, and if so, runs the function that was specified in the configuration. Otherwise it clears the interval and sets a new one. Currently the time for the interval is specified to be 350 milliseconds.

Without this there was a problem with the MathJax matrix that needed to be updated when the student changes the parameters for a transformation. The update of the MathJax matrix slowed down the browser considerably, when it was called more often.

3.3.2.1.7 MaterialSidebar

Module binds click listeners to the left sidebar in the materials page. Data attributes of the elements are read, to determine the correct URL-s to send Ajax requests to.

First listener is for the headings of different panels in the sidebar. Those headings change, which module is opened. When the student opens another module, a request is sent that changes the current course module for the student. This means that if a student refreshes the page or comes back in the next session, the correct module will be opened in the sidebar.

Second, the topic links are listened to. When the student clicks on the topic, a loading bar is shown and a request is sent to fetch the contents. If at that point, student clicks on another topic, then the previous request is aborted and a new one is created. After the topic material is loaded, the loading image is hidden again, and the sublinks for the materials under that topic in the menu are shown, others are hidden. Also the *Examples* module is initialized and cleared during this process.

Third listener is for the flashcards link. When other links actually have the listener disabled, if they are already opened (do prevent multiple requests, if users double-click on the links), then for the flashcards the listener will be enabled again, after the flashcards have loaded. This is because, if the student goes through the flashcards deck and had some cards that were left in the current deck, then the deck can be reloaded, by pressing the flashcards link in the menu.

3.3.2.1.8 Matrix

This module replaces the Latex of a matrix in a specific container by a new Latex containing the representation of a new THREE.Matrix4. Because Three.js holds their matrices in a row-major format, the received matrix is also transposed prior to constructing the Latex. This is because, in the course we are teaching the column-major representation.

There is also a check in here that finds if already an update for the same matrix was queued and not yet parsed by MathJax. This is achieved, by setting a specific class for the container and queuing another job for MathJax after the matrix update. That next job will remove the indicator class. This is because, we are actually modifying a buffer element, not the container which already holds a rendered matrix. When MathJax renders the buffer element, only then, the buffer is swapped against the real container. This is the double-buffering pattern, often used in rendering and it avoids the problem of showing non-rendered Latex, whenever we change the matrix.

3.3.2.1.9 PauseUtils

The *PauseUtils* module checks if an example has an attribute *canPause*. If it is there, then this module adds another button in the top right, next to the full screen button. That other button is the pause button, and it allows the example to be paused, which means that rendering of it is no longer called.

It also binds listeners to the controls of the example. When those are used, then the example becomes unpaused, otherwise a result of the change would not be visible. This could be further developed to render only one frame, when the controls are changed. It would, however, needs some refactoring to have a separate counter (which could also be paused) for the ticks in the examples.

The pause button is very useful for the ray tracing and path tracing examples. Those examples actually start in the paused mode, because of their performance.

3.3.2.1.10 PointUtils

The *PointUtils* module allows to create draggable points in the example. It works with examples that have an orthographic camera that looks at the xy-plane and the points will be located at $z = 0.5$. Dragging occurs with a ray cast into the scene, to determine a point under the mouse. When a point is determined, it will be assigned as a *hoverPoint* to the example. When the user clicks on the point, then it will be assigned as a *clickPoint*.

It also assigns different colors for the point, depending on the states. The usual color of the point is assigned as the drag color. The hover color and drag color are respectively 0.85 and 0.7 times of that color.

This module is used for the examples under the Curves topic. Those examples have the control points and parameters of the curve as draggable points in the example. This allows direct modification of the curve parameters by the students.

3.3.2.1.11 QuickGrader

This module is used in the teacher's pages, specifically in the submissions list. In the submissions list, there are buttons to open the submission to see the details, and for quick grading. When the teacher clicks on the *Quick Grade* button, then a select and a textarea are opened right under the current row. This allows to quickly assign the score and a comment for the submission.

3.3.2.1.12 RendererFactory

Three.js renderes for the examples are created from this module. This is because there is a limited number of WebGL contexts that can be simultaneously active during one session, so renderers are re-used for all the examples.

There is a dynamic pool that collects the renderers that were created, but are no longer used. When a new renderer is needed, then it is first popped from that pool. If the pool is empty, then a new instance is created. There is currently no limit to the size of the pool, but the limit of active WebGL contexts limits the number of examples one material can have. That hard limit is specific to the GPU and browser that is being used.

3.3.2.1.13 Tasks

This module is responsible for the task tree in the Tasks page. It allows users to open and close different modules. Also, each task has a Fancybox modal popup assigned to it, so the tasks would open in the same page. The modal is configured to use an iframe to open the task details, because that view is also used to submit solutions to the tasks. Submitting requires the page to reload and a new request to be sent, thus only the iframe in the modal will do that. Of course, there are ways to send files via Ajax, but for simplicity that functionality is currently implemented that way.

3.3.2.1.14 TaskStats

The *TaskStats* module can receive the values for the task titles, the score, difficulty and time estimates for the current user, the average and the extremes. After receiving those values, it initializes HighCharts plots in specific containers for the score, difficulty and time estimate.

Student's own score is represented with a line chart that has a filled white marker in the point values. Extremes are shown as an area range chart without any markers. Averages are again a line chart, but do not have markers either.

3.3.2.1.15 Utils

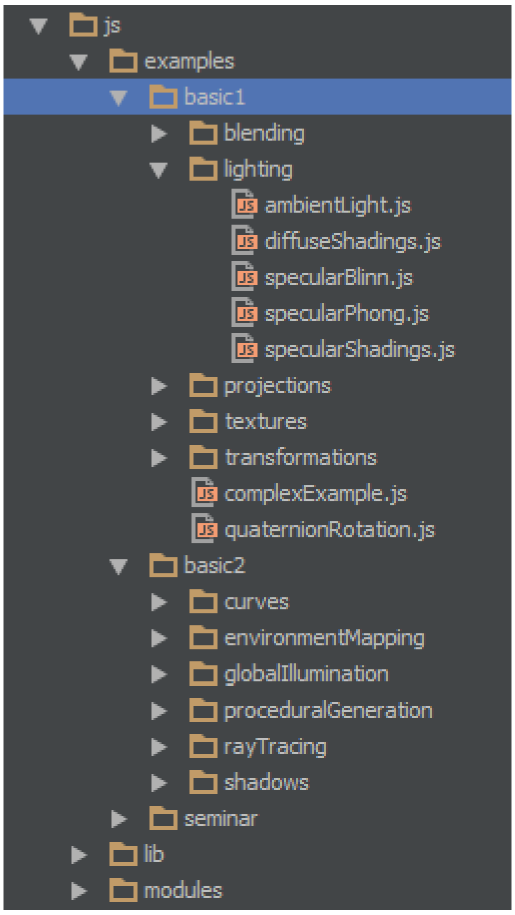
This module groups some general functions that are used in several other modules. Those currently include:

- *millis()* – Uses the JavaScript's Date object to return a number of milliseconds from the year 1970.
- *fixed(value)* – Converts the value into a fixed precision format.
- *toRad(value)* – Converts the value, interpreted as degrees, into radians.

3.3.2.2 Examples

Interactive examples that accompany the material form a big part of the JavaScript in CGLearn (Illustration 6). All the examples follow a similar structure that allows the Examples module to load, initialize and render them.

Usual dependencies for the examples include jQuery for manipulating the HTML DOM, when adding the controls; Three.js for creating the graphical part of the example; *Utils* module, to use the *millis()* and *toRad()* methods. *LiveSlider* module to include sliders with the controls of the example. Specific shader files that are loaded with the help of Require.js text loading plugin [50]. Examples that show a transformation matrix, also depend on the *Matrix* module. Some examples that have Latex near their controls, require the MathJax.


```
▼ js
  ▼ examples
    ▼ basic1
      ▶ blending
      ▼ lighting
          ambientLight.js
          diffuseShadings.js
          specularBlinn.js
          specularPhong.js
          specularShadings.js
      ▶ projections
      ▶ textures
      ▶ transformations
        complexExample.js
        quaternionRotation.js
    ▼ basic2
      ▶ curves
      ▶ environmentMapping
      ▶ globalIllumination
      ▶ proceduralGeneration
      ▶ rayTracing
      ▶ shadows
    ▶ seminar
  ▶ lib
  ▶ modules
```
Illustration 6: File tree of the examples.

Example's constructor is responsible for creating the meshes that will be shown in the example; configuring or creating the camera; assigning different properties that can be used later in the instance. Constructor gets the *exampleContainer* parameter that holds the HTML element surrounding the example. This is usually assigned as a property of the instance and used later, when adding the controls for the example.

The mandatory functions that an example needs to implement, include the *addControls()* and *update(renderer, scene, camera)* methods. First one is responsible for adding different controls under the rendering area of the example and binding listeners to them. Second one is used to either modify the transformations each frame, or render secondary passes.

Other properties include *canPause*, *isPaused*, *fullscreenModes* that configure the corresponding modes of the example.

Most of the examples in one topic generally include the same basic structure, with just different specific calculations applied. Controls usually modify the calculation parameters to show the behavior of a certain algorithm. Next the thesis describes main approaches of the examples in different topics.

3.3.2.2.1 Basic I

Here the implementation details of examples created for the Basic I module are described.

Geometry and Transformations

Examples here specify a *THREE.Line* object with a *THREE.LineMaterial* that has a white color and line width of 1. Depending on the example if may be a box or a triangle.

Controls generally modify the position, rotation, scale properties of the object. In the case of shear, the elements in the transformation matrix are modified directly.

Those examples use the *Matrix* module to display the model matrix of the object.

Shading and Lighting

Although, Three.js has shaders for Phong lighting and allows to specify either smooth (Phong) or flat shading, there is no Blinn lighting model and Gouraud shading available. Generally the examples here include custom shaders that implement all of the required lighting and shading models.

Frames of Reference and Projection

For the orthographic and perspective projection, the corresponding Three.js cameras are used. In the case of the oblique projection, an orthographic camera is used and the projection matrix of the camera changed accordingly.

Examples here show the current projection matrix under the rendering area.

Textures and Sampling

The examples with a textured cube and a texture quad load the textures in the usual way with a *THREE.ImageUtils.loadTexture()* method. The textures are also mapped to the objects in a standard fashion. The offset parameter, in the case of the checkerboard texture rendered on a quad, is increased each frame to create the movement of the texture.

The sampling example that shows the sampling of a function with ever increasing frequency, uses a custom shader that samples that function.

Blending

The example in this topic was implemented by Jaanus Jaggo. It uses custom shaders to reverse the projection transformation from the depth values, in order to show the linear depth values for each fragment.

3.3.2.2.2 Basic II

Here the implementation details of examples created for the Basic II module are described.

Environment Mapping

The cube map example uses the *THREE.CubeCamera* object to render the scene from inside the central mesh. The sphere map example also uses the *THREE.CubeCamera* to create an initial reflective sphere in the object. Then it renders that scene with an orthographic camera, to generate a sphere map. Custom shaders are used to sample the reflected values from that sphere map.

Curves

Examples here include the *PointUtils* module to create interactable (draggable) points inside the examples. The interpolating cubic curve, Hermite and Cardinal spline examples use the predetermined blending functions for the curve. Bezier spline example constructs the curves via the geometric interpretation of the de Casteljau's algorithm, by recursively dividing the segments at a given $t \in [0..1]$. B-Spline and NURBS examples construct the blending functions recursively directly by the de Casteljau's algorithm using the knot vector.

Procedural Generation

The Perlin noise example sends to the custom shaders a texture with random monochrome values in it. In the fragment shader that texture is sampled 6 times with a specified level of zoom. Those 6 samples are then mixed together with specified coefficients. This algorithm is described in [51].

The Lindenmayer-system example iterates the system, based on 4 rules, the number of times specified. The probability of each rule can also be specified. After iterating the system enough times, the resulting word is then converted to a hierarchical line mesh, where each line has the red color.

Particle systems and Boids example use a *THREE.PointCloud* object to define the particle mesh. The Boids algorithm is an adaptation based on the pseudo-code in [52].

Ray Tracing, Space Partitioning, BVH

The ray trace rendering example creates a data texture to send the geometry information into the custom shaders. For each fragment the geometry information is read, to test the intersections with all of the triangles in the geometry. The Möller-Trumbore ray-triangle intersection algorithm is used that. The nearest intersection is found. If that nearest intersection belongs to the reflective sphere, then another ray is cast, to determine the color value of the fragment. If a ray hits the light source, then the color white is returned. Otherwise the diffuse light is calculated, based on a normal, also sampled from the data texture.

Global Illumination

Path tracing example follows a similar structure, as the ray trace renderer example from the previous topic. Difference is that a number of bounces is specified (when that is changed, new shaders are compiled, those have a different number of bounces in them). At each intersection, where a ray has not reached the number of bounces, a random direction from the hemisphere surrounding the surface normal is found. This is done, by generating a random vector [53] and testing if it lies inside the hemisphere around the normal. If it does, it is normalized and returned. If it is not, then another random direction is tried, this goes on for 15 times, after which the function returns the last value. GLSL does not allow infinite loops and within 15 times it is sufficiently probable that a suitable vector is found.

The random ray is traced recursively until the number of bounces is reached or the ray intersects a light source. At each hit with the diffuse geometry, a diffuse reflection is calculated. Here no ray is traced to the light source, because this example should not include shadows.

Shadows

The global illumination shadows example has the path tracer from the previous topic extended to also shoot shadow rays towards the light source. There are 4 shadow rays shot to 4 random positions on the area light source. Based on the number of shadow rays hit, the diffuse light reflection is taken into account with less weight. If all 4 rays hit, then no diffuse light is considered, each miss increases the consideration by 25%.

Although Three.js has shadow mapping implemented in the library, it only works for directional and spot lights. In order to have a point light in the center of the scene that casts shadows with shadow mapping, a cube camera was used to render 6 shadow maps. The resolution of that cube camera can be changed via the controls. The cube map is then sent to a custom shader that gets sampled for each fragment by its world coordinates (because our light source is in the center of the world). For each of the cameras belonging to the cube camera, the projection matrix is multiplied with the view matrix, the result is sent to the shaders for normal rendering. During the latter, each fragment has its position found in the spaces of all 6 cameras. If the position is inside the view volume of the specific camera, the depth value is saved and compared with the value in the cube map.

Three.js removed their shadow volume implementation in r43 in favor of shadow mapping [54]. The shadow volume algorithm was implemented by changing the projection matrix to have a far plane at infinity. The edges of all meshes were found and silhouette edges are detected each frame. The shadow volume consists of three parts: light/front cap, dark/back cap, sides. In order to construct those meshes, the vertices of the original object are duplicated and scaled a bit down. The vertices belonging to the silhouette, are duplicated again and faces between both of them, are constructed. After this, the duplicated vertices of the silhouette and the dark cap vertices are extruded to infinity. Because Three.js sends the vertex coordinates as triplets, another attribute was added to specify the *w*. When rendering, the depth-pass version of the shadow volume algorithm was used.

3.4. Functionality

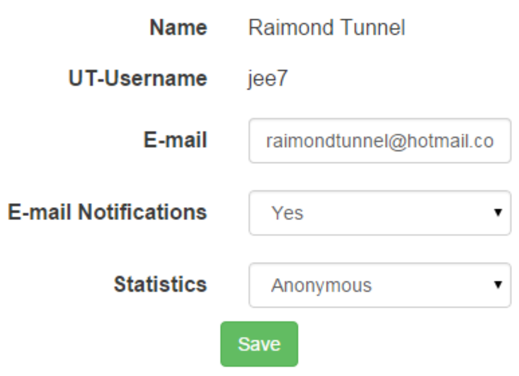
This section describes the current functionality of the system. Besides giving an overview of the features, it also serves as a general user manual.

3.4.1. Student

Students are able to log in, while authenticating themselves via the University's authentication system that uses Shibboleth. The Service Provider part of Shibboleth was configured for CGLearn and use of it confirmed with University's IT department.

Shibboleth gives the basic information about the students: their name, e-mail, role in the university. Unfortunately, it does not give the student number, nor tells whether the students are registered to this course or not.

Students have a profile page, where they can change their e-mail, to which grade notifications and feedback is sent to. They can also disable the e-mails sent to them by the system (Illustration 7).

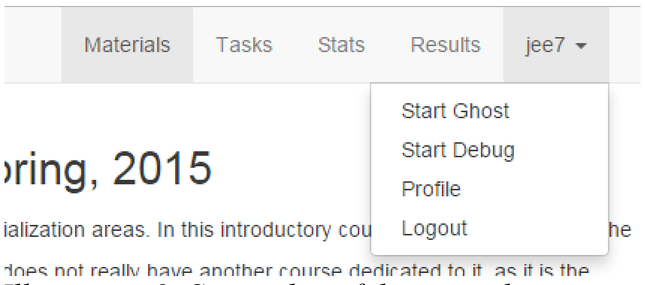


Name	Raimond Tunnel
UT-Username	jee7
E-mail	raimondtunnel@hotmail.co
E-mail Notifications	Yes
Statistics	Anonymous

Illustration 7: Screenshot of the profile edit view.

Everyone, who has authenticated themselves, can view all the material of the active courses. Currently, there are two active courses in the system: the Computer Graphics (spring 2015, MTAT.03.015) course and the Computer Graphics Seminar (spring 2015, MTAT.03.305) course. The dropdown in the upper left corner allows to switch between them.

In the right corner, there are links to the pages inside the system: Material, Tasks, Results, Stats. The last link shows the username and allows the user to go to their profile page or log out (Illustration 8).



Materials
Tasks
Stats
Results
jee7
Start Ghost
Start Debug
Profile
Logout
›ring, 2015
ialization areas. In this introductory cou
he
does not really have another course dedicated to it, as it is the

Illustration 8: Screenshot of the top right corner. Students do not have the Start Ghost and Start Debug links.

3.4.1.1 Material

Material of the course is navigated via the left sidebar that groups different topics into modules. The modules can be closed and opened in the sidebar. The opened module shows the list of topics in that module. When a topic is opened, a list of materials under that topic also appears (Illustration 9). The last module opened, is remembered by the system as the current course module.

The opened material starts with the topic's title and introduction (description). Usually the introduction ends with a couple of definitions that the student should pay attention to in that material.

Next come the different materials under that topic. They have a title and the content. Often the content of the materials (as well as the topic itself) includes illustrations. Occasionally there is an interactive example inside the material, those are aligned to right and come in various lengths. The rendering area and the width of the example is always the same.

Students can scroll down to read the material, interact with the examples inside the material. They can also navigate the material using material links under the opened topic in the right sidebar. When they want to switch the material, they can click on another topic or open another module.

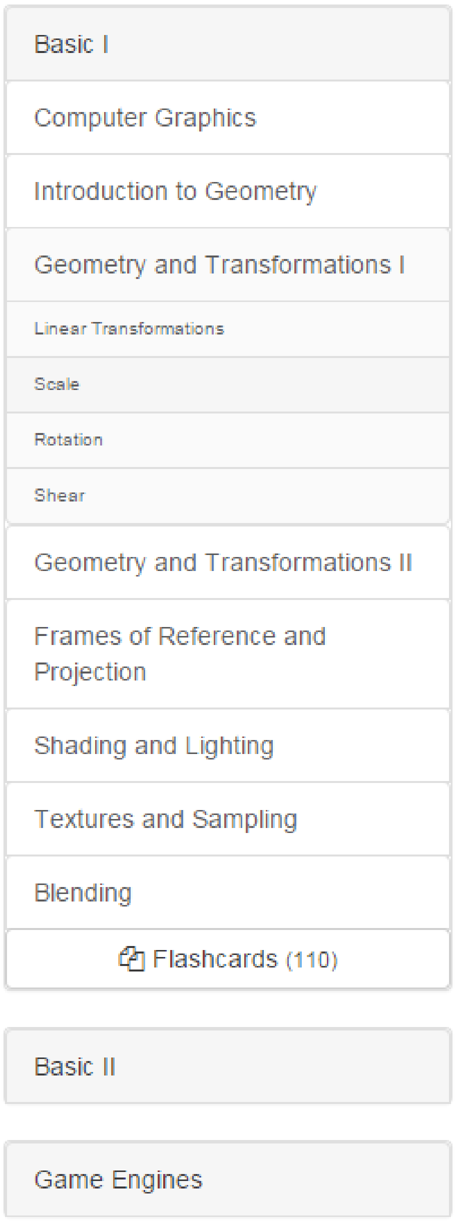


Illustration 9: Left sidebar. Basic I module and the topic Geometry and Transformations I is opened. Materials under that topic are Linear Transformations, Scale, Rotation and Shear. Basic II and Game Engines modules are currently closed.

3.4.1.2 Tasks

The tasks page shows the tasks tree. The first hierarchy is based on the hierarchy of modules. Basic I is the root node and Basic II and Game Engines modules are the child nodes. Each of those nodes can be opened by clicking on the title. Only one module in the same level can be opened at a time. This means that when Basic II is opened, Game Engines is closed and vice versa.

Opened module shows a brief description of that module and the instructors responsible for grading and instructing the tasks. Tasks are then laid out one topic at a time. Inside one topic, there can be tasks in multiple rows, depending on the dependencies of tasks. Tasks with preceding requirements are in one row.

Each task has also the task number in the top left corner of the box and the average time spent by the students in the top left corner. In the center is the task's title, and under that there is a progress bar showing the amount of points earned for that task.

If the progress bar is red, then this task has not been submitted, but could be if the student solves it. If the progress bar is yellow, then the task is submitted, but not yet graded by an instructor. Green progress bar indicates that the score is given and displays the amount of points received by the student. Blue progress bars are under tasks that the current student has missing prerequisites for, *i.e.* they have not submitted a solution for the preceding task yet (Illustration 85 and Illustration 86 in the Appendix).

When a task is opened, a modal popup is displayed with task's description and a form to submit a solution (Illustration 87 in the Appendix). Task description generally includes the overview of the technique required and the goal postulated for the task. Students have the ability to submit multiple solutions with corrections (even after initial points and feedback is given). Different texts show in what status the submissions are. The feedback and discussion between the student and the instructor is also shown above that form (Illustration 88 in the Appendix).

3.4.1.3 Statistics

The statistics page shows three plots. The first plot shows the average score, minimum and maximum scores, and the score of the current student for all the current tasks in the course (Illustration 89 in the Appendix). The second shows the same indicators for the difficulty estimations (Illustration 90 in the Appendix) and the third one is for the time estimations (Illustration 91 in the Appendix).

3.4.1.4 Results

Results table shows the overall table of scores from both the different modules and from the specific gradable results (project and the exam) in the course. The names in the table are fake and rows gray (Illustration 92 in the Appendix). The current student has a black row and a real name. Overall total and a grade based on that total are shown in the last columns.

3.4.2. Teacher

Users that are given the rights of a teacher, can access the teacher's part of CGLearn. From there they can edit different parts of the material, assign scores and feedback for the submitted solutions, import students to a course, go into *ghost mode* of a specific student or view a complete results table. The different functionality can be accessed by a menu in the top right corner (Illustration 10).

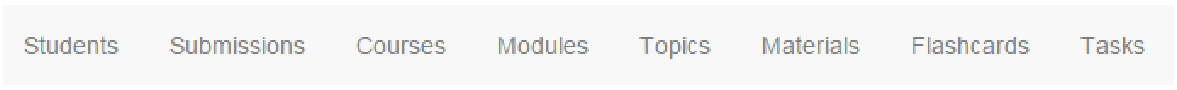


Illustration 10: Top right menu in the teacher module.

3.4.2.1 Material

Editing any textual part of the material is done via a TinyMCE editor field. There are different listings of the specific parts of the material (courses, modules, topics, materials, flashcards, tasks) that allow editing (Illustration 93 in the Appendix). The editor itself is configured with the basic buttons to force a similar style of the material (Illustration 94 in the Appendix). More advanced options would create a lot of overhead for the teacher editing the material. Specific elements or styles can also be added by editing the source HTML directly (TinyMCE provides a tool in the Tools menu for that).

Adding an example in the material consists of first creating the example in JavaScript and GLSL, after which the example needs to be added to the Git repository. To add a finished example to the material, a <div> element with a specific style needs to be created. There is a button in the Tools menu for that as well.

Adding images to the material works in a similar fashion. The support for image and file upload is not configured in TinyMCE, because all the teachers have access to the Git repository, thus uploading it via browser will take them the same amount of time and effort.

3.4.2.2 Submissions

In the submissions view it is important to show only the submissions that need a response from the teacher. There is also an additional filter available to filter by topic. That filter also shows how many new submissions there are in each topic. On the right there is a Bulk Download button (Illustration 95 in the Appendix). Clicking that makes the server repack all the new solutions into another archive that has a well structured folder system. The first level has the tasks and the second level has the students, who have submitted solutions for the task. Depending on the number of pending tasks, this process may take some time for the server.

Each submission can also be individually downloaded. On each row there is the essential information about the submission. Part of the last comment from the student is also shown there. Each row has a button for opening the solution in another view, to see the feedback and task description in more detail. Alternatively, the teacher can click the Quick Grade button, and assign a grade and a comment right then and there (Illustration 96 in the Appendix). That data is sent via Ajax, thus the teacher can continue grading other solutions without the page refreshing.

3.4.2.3 Results

For teachers, the entire results table is shown with all the correct names of the students (Illustration 97 in the Appendix). That table also allows the teacher to assign points for additional gradable results associated with that course. Currently these include the project and the exam. This table can be accessed by a blue *Results* button under the course list.

3.4.2.4 Student Import

In order to associate students with a course in CGLearn, an import functionality is implemented. Associating every student by hand would take too much time from the teacher. Instead, a list of students can be exported from the Study Information System (SIS) and the result used to search CGLearn for the corresponding users. That import functionality tries to match students by their name (if this would be ambiguous for some name, or no such student is found), then it will try by using the two e-mails also provided by SIS.

The result of the import shows how many students were found, how many were not found, and how many were actually new associations. This also means that the teacher can import the same CSV multiple times and only the new associations are created.

4. The Material

The created material for the Computer Graphics course is organized in 4 layers of hierarchy (Illustration 11). The root node is the course, under which are the modules. The modules group under them a number of topics with a common larger goal (*e.g.* "get the basic understanding of the essential computer graphics techniques").

Each topic covers a smaller aspect of that goal. Examples include a topic for texturing and sampling, or a topic for shadows. Topics also have a description, which explains the main ideas covered by the topic.

Under the topics, there are smaller sections called *materials*, those focus on a single technique or concept. Inside a material there are descriptions, illustrations, formulas and interactive examples.

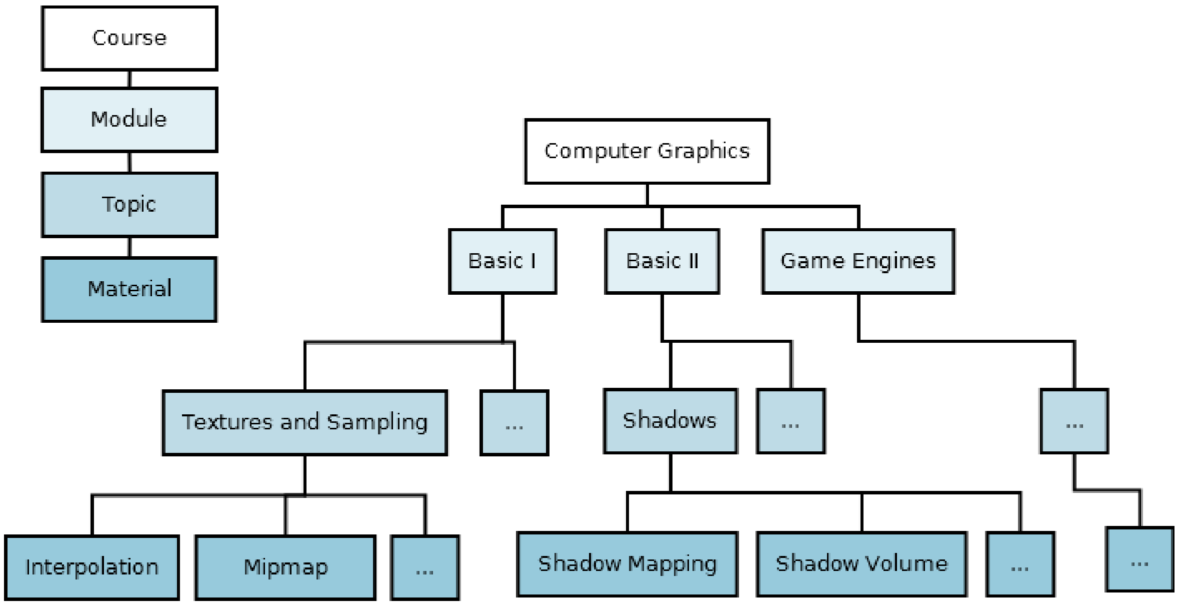


Illustration 11: Overview of the hierarchy of materials.

Part of this thesis work was to create most of the material for the topics under Basic I and Basic II modules. Two topics were written by Jaanus Jaggo and Timo Kallaste under those modules. The Game Engines module consists entirely of the material written by Ats Kurvet and Timo Kallaste.

The aim of the material was to provide a clear, interesting and familiar source of information on the topics covered in the modules under the course. Online material that follows the course structure is an important aspect of learning. Interactive examples should consolidate the students' understanding of the material. This is because, they allow students to directly manipulate the covered ideas and techniques – they can see the results right away, and do not have to imagine them. This corresponds to level 4 engagement (changing) in the engagement taxonomy proposed by Naps et al [15].

The material is written partially in a problem-oriented approach. This means that a problem statement (*e.g.* "we want to specify a more granular surface color") is first postulated. Then some logical approaches towards solving the problem are made. Depending on the problem, the approach may highlight specific properties or drawbacks of an algorithm (*e.g.* as it is with shadow mapping) or focus on a rigorous formulation (*e.g.* material under the Curves topic).

Topic introductions list a number of key definitions that the students should pay attention to. Usually the definitions are prerequisites for understanding the subsequent material.

Different new terms in the material, which are also explained, are linked to corresponding Wikipedia articles for further reading. This is important so that students can form an extensive network of ideas and relate the material to other sources. Often there are also links (references) to more thorough external sources that cover the material from another perspective or in more detail. Most of those external sources are also mentioned in the forthcoming text of this thesis.

4.1. Basic I

Basic I module covers the most basic topics in computer graphics. Those focus on the different transformations and light computations. The aim of this is to introduce to students the essentials, without which doing standard 3D graphics would be impossible. This module should also allow students to understand that a large part of 3D graphics consists of trigonometry, geometry and algebra, which they have studied in previous courses like Algebra and Geometry (MTMM.00.271), Algebra I (MTMM.00.307).

A strong foundation in those topics gives a better understanding of more advanced topics. For example, in Basic II the topic of ray tracing requires students to have a good understanding of 3D geometry; the topic of shadows will enhance the projection matrix and requires students to understand projection.

4.1.1. Computer Graphics

The introductory topic titled "Computer Graphics" will give a general overview of the areas, where computer graphics is applied. It talks about the layout of graphics programming using API-s like OpenGL and WebGL. Furthermore, it includes an introduction to the affine Euclidean space, which will be needed in the geometry topics.

4.1.1.1 Introduction

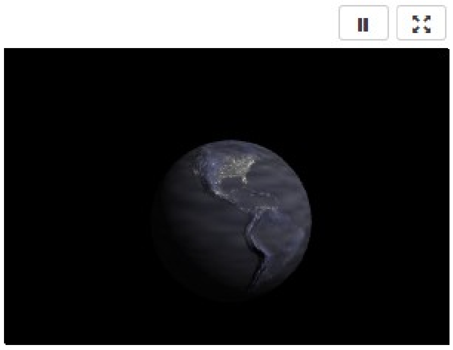


Illustration 12: Example with a texture,. bump mapping, blending, noise generation, rotation and curve following.

We start by describing the role of graphics libraries, graphics API-s and the GPU. This is done in the context of a JavaScript application. An example of a graphics library is Three.js. An example of a graphics API is WebGL. It is described how the library uses the API to send calculations to the GPU.

The material ends with an example of a rotating 3D approximation of a sphere that has the night map of the Earth as a texture, bump mapping based on the texture and rotating clouds based on Perlin noise blended to the texture (Illustration 12). The object also moves along a predefined curve. That example is briefly described in the material and should give a good overview, what can be achieved with mostly the techniques in Basic I.

4.1.1.2 Technologies

Because the course focuses on both a C++ and JavaScript application, this material gives a comparison between them.

We introduce the notion of a C++ application that has the code executed on a relatively low level, which makes it an ideal choice for computation-heavy real time applications such as computer games. C++ can use OpenGL API to communicate with the graphics card and there exists a 2D graphics library Allegro, which is often used to make 2D games.

The following material introduces JavaScript, which is usually run in a web browser and thus is not as efficient as compiled C++ code. On the other hand, the Web allows other people to run the code without the need to specifically download it or compile it themselves. Hence, this would serve another target audience and graphics on the web can be done with WebGL API and optionally the use of Three.js graphics library.

4.1.1.3 Related Math

This material could be quite long and some sources, such as [55], have an entire chapter dedicated to it. To keep the material concise and related for the students, it contains only an overview of the affine Euclidean space, explains the difference between position and direction vectors and shows how to create shapes by taking linear combinations of position vectors. Most notably, it is shown how a convex combination of 4 specific position vectors can define a square.

4.1.2. Introduction to Geometry

This topic delves more into the notion of coordinate systems and vectors. It lays out the various notions in the field of geometry that are most important and essential for computer graphics. Lastly, it describes what properties different polygons can have and how can you represent polygons with only triangles.

In the end of the description of this topic several definitions are stated. This is a recurring theme for most of the topics and should give students a clear overview, which terms will be important and how do they relate to the following material.

4.1.2.1 Coordinate Systems

Here the material shows that the directions of the basis of a coordinate system need to be agreed upon. It is shown that the idea of *a character moving 3 units left and 4 units up*, may be quite ambiguous, if we have not agreed upon our coordinate system handedness and what directions different basis vectors actually denote.

Together with a left-handed and right-handed coordinate systems we show that the positive direction of an angle also depends on the handedness. This is an important concept that should be considered each time there will be calculations with angles in the latter material.

To further connect the student's previous understanding with the current material and to refresh the geometric thinking, the Gram-Schmidt orthogonalization process is also briefly shown in this material.

4.1.2.2 Points and Vectors

This material shows the difference between position vectors (that in computer graphics are called points) and direction vectors (that in computer graphics are called just vectors). It mentions the importance of distinguishing between the two and states how homogeneous coordinates are used to do that.

4.1.2.3 Polygons

Generally to render a meaningful shape in computer graphics, it should be constructed from polygons (with a few exceptions). The material shows that the order of the points can be specified to uniquely fix a certain polygon. Depending on the order and the actual points, we can get simple or non-simple polygons. Simple polygons can be considered to be convex or concave. Convex polygons should relate to the convex combination from the previous topic and it is explained that triangles, being the most simple polygons, will always be convex.

The material also explains that polygons can have a front face and a back face. This is important from the graphics perspective, because it allows us to effectively ignore the back faces of polygons without usually sacrificing the visual result.

Lastly, the triangle strip and the triangle fan are shown. It is told that these are often used to construct larger polygons out of triangles.

4.1.3. Geometry and Transformations I

The topic starts with an example that shows how to scale a triangle with respect to the origin by two. It provides the three vertices of the triangle in a 2D coordinate system and asks the student to think what needs to be done to scale the shape. It is shown that multiplying both of the coordinates with 2, the new coordinates would form a scaled shape. Then it follows that multiplication with a particular matrix will achieve that transformation.

After this it is mentioned that one could also think about the basis vectors and how the transformation would affect those. When doing transformation it is often useful to think of the transformation applied to the basis vectors, instead of a more complex geometric object.

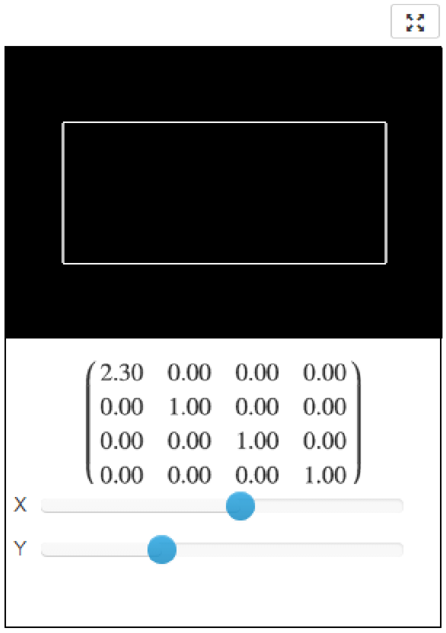
4.1.3.1 *Linear Transformations*

Because this topic covers only the linear transformations (that do not need homogeneous coordinates), the notion of linearity of a transformation is defined and proved. It is shown that all matrices with fixed elements will define a linear transformation on a vector space. This should give students an understanding that there can be as many different transformations, as there are different matrices.

4.1.3.2 *Scale*

Next, the scale transformation, that was also part of the example in the topic's introduction, is defined and shown. The coefficients in the matrix can either shrink, enlarge or keep scale the same for the different coordinates / basis vector directions.

This material has the first transformation example accompanying it (Illustration 13). It presents a square that can be transformed with the scale transformation. The example has two sliders underneath it, which allow the user to change the scale coefficients for x and y in the matrix. The matrix presented is a 4×4 matrix that has the 2D linear transformation in the top-left 2×2 part. The full affine transformation matrix for a 3D space (4×4) is shown, because later examples will modify the different parts of it. This provides students already a general way of thinking about the transformations, without the need to build the full matrix step-by-step.



$$\begin{pmatrix} 2.30 & 0.00 & 0.00 & 0.00 \\ 0.00 & 1.00 & 0.00 & 0.00 \\ 0.00 & 0.00 & 1.00 & 0.00 \\ 0.00 & 0.00 & 0.00 & 1.00 \end{pmatrix}$$

Illustration 13: Scale transformation example with the full affine transformation matrix shown. Example has sliders for the x and y scale coefficients.

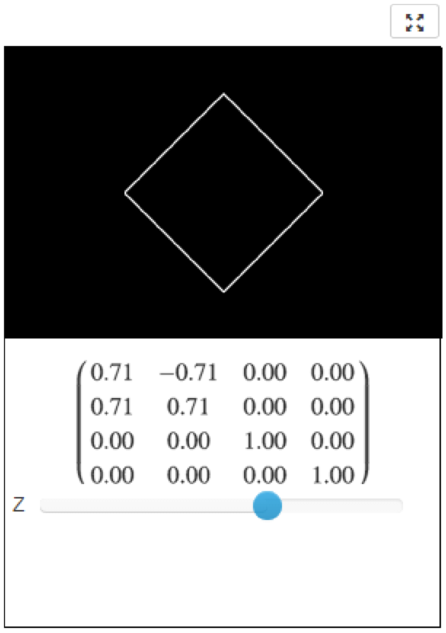
4.1.3.3 Rotation

The scale transformation is followed by the rotation transformation, because it shows another common linear transformation. Material could introduce the shear transformation before rotation, but shear may not be that relatable to students at first.

For rotation, the specific matrix is derived by rotating the standard basis vectors by an angle α. This should give students an understanding how rotation is achieved using trigonometry. Otherwise the nature of the rotation matrix may become a missing link for the students.

It is specified that rotation is a linear transformation only when we fix the angle α. That will cause the elements of the matrix to become specific real numbers, thus the proof of linearity has the satisfying conditions.

Rotation is also accompanied by a example, where students can change the angle α for a rotation of a 2D square around the z axis (Illustration 14).



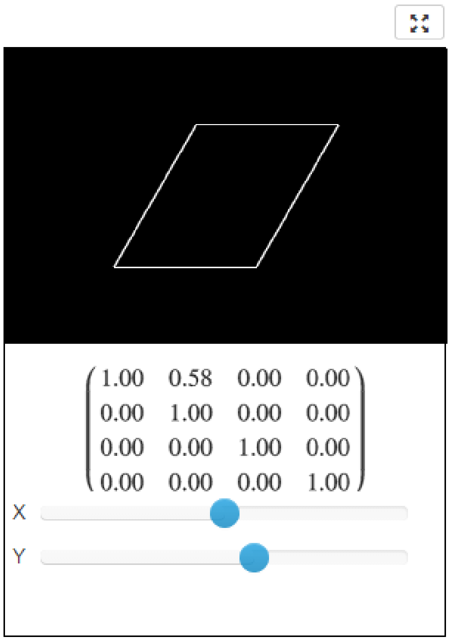
$$\begin{pmatrix} 0.71 & -0.71 & 0.00 & 0.00 \\ 0.71 & 0.71 & 0.00 & 0.00 \\ 0.00 & 0.00 & 1.00 & 0.00 \\ 0.00 & 0.00 & 0.00 & 1.00 \end{pmatrix}$$

Illustration 14: Rotation transformation example that has a slider for rotating the square around the z axis.

4.1.3.4 Shear

Lastly, the shear transformation is introduced. It is mentioned that this transformation will find a use later in the material. At this point students should acknowledge that such a transformation exists.

The example allows the student to do the shear-x and shear-y transformations in 2D by specifying the angle ϕ by which a corresponding axis gets tilted.



$$\begin{pmatrix} 1.00 & 0.58 & 0.00 & 0.00 \\ 0.00 & 1.00 & 0.00 & 0.00 \\ 0.00 & 0.00 & 1.00 & 0.00 \\ 0.00 & 0.00 & 0.00 & 1.00 \end{pmatrix}$$

Illustration 15: Shear transformation example. Allows students to specify ϕ for shear-x and shear-y transformations.

4.1.4. Geometry and Transformations II

This topic will extend the linear transformations to affine transformations to include the translation. It also covers the notion that the order, in which transformations are applied matters and affects the final result. Lastly, it shows a *scene graph* and explains how a stack of matrices can be used to save and load transformation matrices, when traversing through the scene graph.

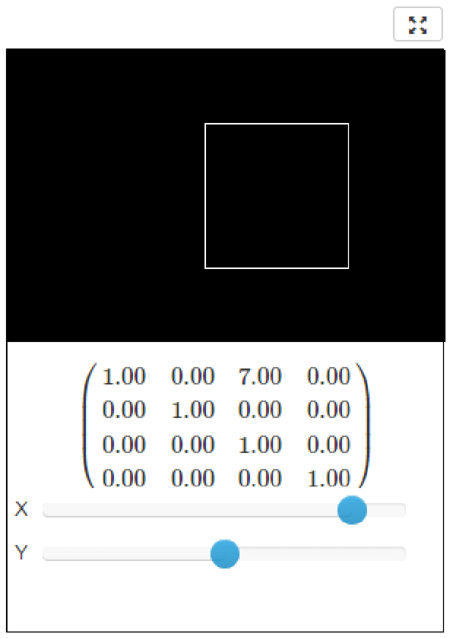
The introduction starts by explaining that our goal is to move our geometry away from the origin, to some specified locations in the space. When doing that, the transformations still need to be applied in the correct order. This should give students an initial goal, to which the material provides answers to.

4.1.4.1 Translation

The material starts with a simple example of a 1D world located on the $y=1$ line. Students are faced with a problem to translate all of the objects in that world. This means that the objects should all change their position by a same, fixed amount, regardless of their previous position. They should also stay on the $y=1$ line.

It is shown that if we apply shear-y transformation on the 2D world, then the y coordinate of all the objects stays the same, but x will change by a fixed amount. This is the basis of the construction of the 4×4 augmented affine transformation matrix that has followed all the interactive transformation examples so far. Explanation continues, how a translation in 2D could be a shear-xy in 3D. The translation in 3D will be the shear-xyz in 4D.

The example in this material allows the student to move a 2D square located on the $z=1$ plane, by applying shear-xy on it (Illustration 16).



$$\begin{pmatrix} 1.00 & 0.00 & 7.00 & 0.00 \\ 0.00 & 1.00 & 0.00 & 0.00 \\ 0.00 & 0.00 & 1.00 & 0.00 \\ 0.00 & 0.00 & 0.00 & 1.00 \end{pmatrix}$$

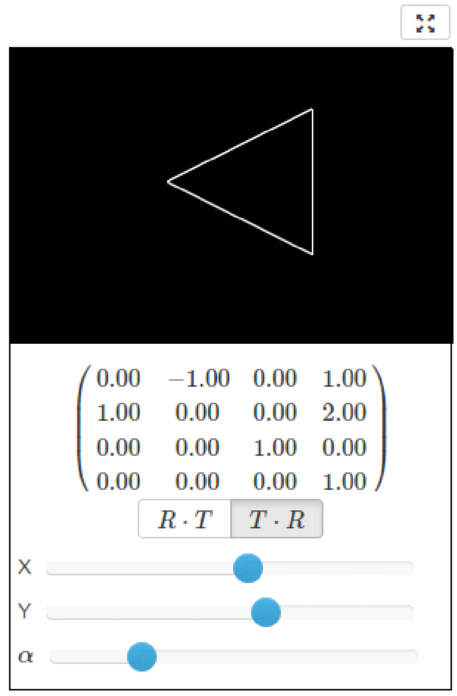
Illustration 16: Translation example that allows student to translate using shear-xy.

4.1.4.2 *Multiple Transformations*

After introducing all the main transformations, it is logical to talk about applying many of them sequentially. First it is shown that because matrix multiplication is commutative, we can multiply all the transformations together and then apply to the vertices. This is the main reason why we use matrices to represent transformations in computer graphics.

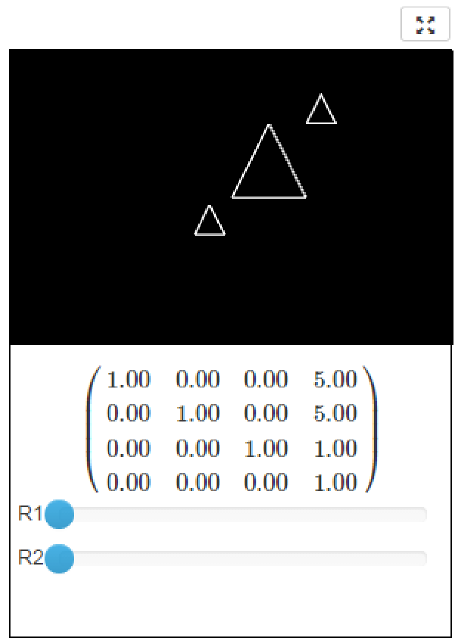
Next, this material shows that extra care should be taken with the order in which the transformations are applied. It is done by an example, where one transformation is rotation by 90° around the z axis, and the second transformation is a translation by $(1, 2, 0)$. Using matrix multiplication, those two transformations produce a different matrix, when multiplied together in a different order. It is emphasized that the transformations are applied from right to left.

The example shows a triangle that has a translation and rotation transformations applied to it. The example allows to specify different x and y translations and an α for the rotation. There are two buttons that specify in which order those transformations are applied. Given the same transformation, the resulting triangle will be different depending on the order.



$$\begin{pmatrix} 0.00 & -1.00 & 0.00 & 1.00 \\ 1.00 & 0.00 & 0.00 & 2.00 \\ 0.00 & 0.00 & 1.00 & 0.00 \\ 0.00 & 0.00 & 0.00 & 1.00 \end{pmatrix}$$

Illustration 17: Multiple transformations example. Order of the specified translation and rotation can be changed.



$$\begin{pmatrix} 1.00 & 0.00 & 0.00 & 5.00 \\ 0.00 & 1.00 & 0.00 & 5.00 \\ 0.00 & 0.00 & 1.00 & 1.00 \\ 0.00 & 0.00 & 0.00 & 1.00 \end{pmatrix}$$

Illustration 18: Example that has the smaller triangles as children of the bigger one. Rotating the bigger one also rotates the smaller ones around it.

4.1.4.3 *Matrix Stack*

Here the material explains that there is usually a dependency between different objects in the scene. This dependency means that some transformations might be the same for some objects, but those objects may have their own extra transformations.

This is illustrated by an example with two triangles located around a third, larger one. Whatever transformation is applied to the third triangle, should also be applied to the first two. The smaller triangles, however, can have their own transformations.

In order to multiply the transformations of one of the smaller triangles to the current transformation, while still keeping a copy of the previous transformation, a stack can be used. In the material there are illustrations that show how to use a stack while traversing the scene graph.

The interactive example shows the recently described two smaller triangles around a third, bigger one. Student can manipulate the rotation of the bigger triangle, which effectively also rotates the smaller ones together with it. Another slider allows to modify only the rotation of the smaller triangles (Illustration 18).

4.1.5. Frames of Reference and Projection

Having covered different transformations, this topic describes the important frames of reference used in computer graphics. There are different key coordinate systems that make up the standard graphics rendering process. Transformation matrices transform the coordinates from one to another. One of the last such transformations is the projection transformation. There are different types of projections, which are also covered together with the matrices for them.

4.1.5.1 Frames of Reference

Frames of reference are described in the order that goes from our 3D space to the actual 2D screen coordinates. The object's local space was already shown in the previous material, where geometry was described around some central origin of that object.

The next logical step is to specify a world coordinate system in which the different objects are located. The world space is described as a space into which objects are transformed with their own modeling transformations.

From the world space, we move on to the camera space. The transformation matrix is derived from the camera's *lookAt*, *up* vectors together with its position. For simplicity we assume that the *up* vector is already orthogonal to the *lookAt* vector. The derivation of the *right* vector includes a reference to the Gram-Schmidt process, although in its simplest form, we are only taking the cross product. This should show the students an application of the cross product. After this, we apply the inverse translation and the inverse camera's model transformation in a reverse order. For the students this should connect with the order of transformations described in the previous topic.

The clip space and the canonical view volume are illustrated for the perspective projection. It is mentioned that for an orthographic projection the initial view volume would be just a cuboid. In order to achieve the canonical view volume for the perspective projection, the homogeneous point normalization needs to be performed. Material mentions that the perspective projection matrix changes the *w* coordinate of the homogeneous points and the GPU does the division itself to get to the canonical view volume.

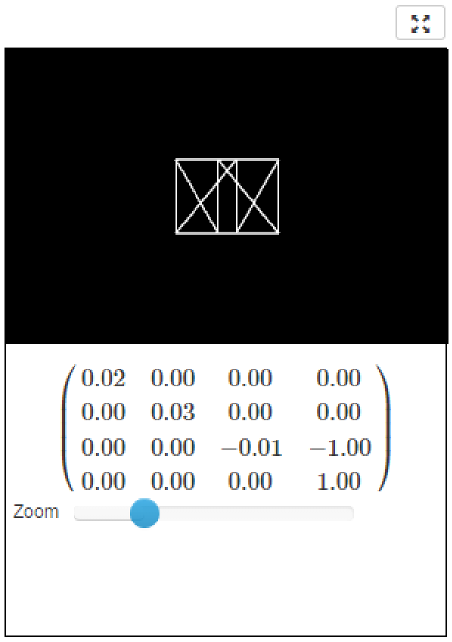
Lastly, a transformation from the canonical view volume into screen space is shown. Although this transformation is performed automatically inside the standard graphics pipeline by the GPU, from the student's perspective it is good to see that it is just a matrix that does scaling and translation.

4.1.5.2 Projections

The material shows three different projections: orthographic, oblique and perspective.

First, the orthographic projection is derived, because it is the simplest. The derivation assumes that the projection is symmetric, because that also simplifies the resulting matrix.

The interactive example shows a rotating wireframe cube projected with an orthographic projection. There is a slider to zoom the cube. It is important for the student to see that no depth perspective is happening with orthographic projection. The front and back faces of the cube are of the same size (Illustration 19).

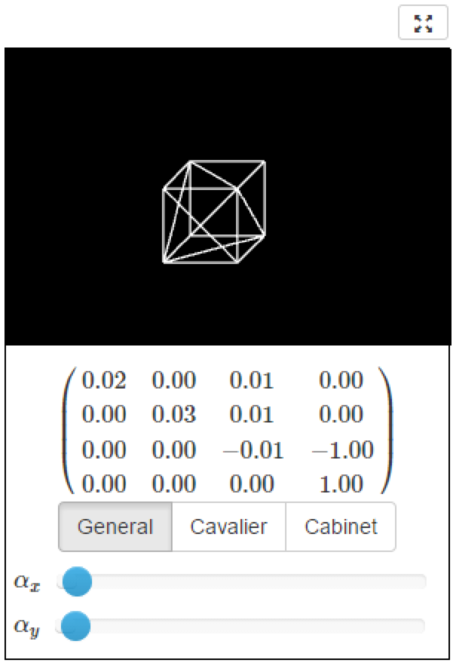


$$\begin{pmatrix} 0.02 & 0.00 & 0.00 & 0.00 \\ 0.00 & 0.03 & 0.00 & 0.00 \\ 0.00 & 0.00 & -0.01 & -1.00 \\ 0.00 & 0.00 & 0.00 & 1.00 \end{pmatrix}$$

Illustration 19: Orthographic projection example.

Secondly, forms of oblique projection are covered. This is because the oblique projection is directly derived from the orthographic projection. Here the general oblique projection is shown, together with the cabinet and cavalier projections. Students should be able to relate this directly with engineering drawing classes from high school or the way people usually draw a cube on any 2D plane.

Interactive example shows a stationary cube and allows the student to modify the two angles in the shear-xy for the general oblique projection. There are also modes for the cabinet and cavalier projections, where the student can modify the one free angle.

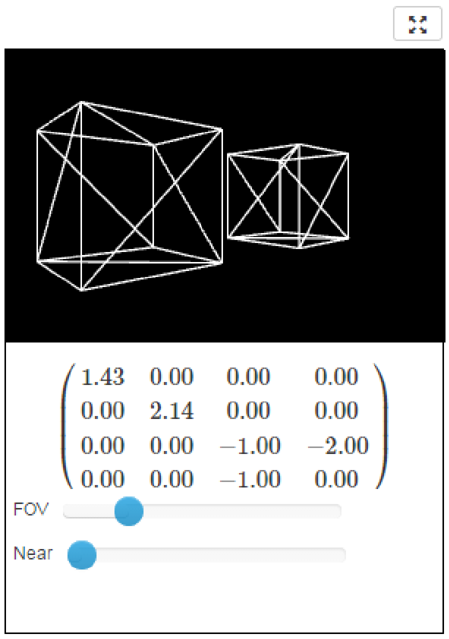


$$\begin{pmatrix} 0.02 & 0.00 & 0.01 & 0.00 \\ 0.00 & 0.03 & 0.01 & 0.00 \\ 0.00 & 0.00 & -0.01 & -1.00 \\ 0.00 & 0.00 & 0.00 & 1.00 \end{pmatrix}$$

Illustration 20: Oblique projection example.

Lastly, the perspective projection is derived from the field-of-view (FOV) parameter and the aspect ratio, as it usually is with perspective projection. The derivation includes similar triangles and shows, why we need to divide every coordinate with -*z*. This should illustrate that we can not represent this as another affine transformation, but have to use the homogeneous coordinate and the point normalization (*w*-division, perspective division).

It is also shown that we need to transform the existing *z* coordinate in order to preserve the depth information. Specific scale and translation values are derived in the material. Side-effect of those is that the depth values will become non-linear. That is illustrated by a table that shows the depth values for a case, when $near=1$ and $far=10$.



$$\begin{pmatrix} 1.43 & 0.00 & 0.00 & 0.00 \\ 0.00 & 2.14 & 0.00 & 0.00 \\ 0.00 & 0.00 & -1.00 & -2.00 \\ 0.00 & 0.00 & -1.00 & 0.00 \end{pmatrix}$$

Illustration 21: Perspective projection example.

The example will show two cubes that rotate around the world origin (Illustration 21). Students can change the FOV or the value of the near plane. This example should illustrate that with perspective projection we actually do get the perspective effect that is natural in the real world. The cube in the front will seem bigger than the cube in the back. Moving the near plane further away, will illustrate that some parts of the cubes may be clipped by it, because they will be closer to the camera than the corresponding value.

4.1.6. Shading and Lighting

With different transformations and frames of references covered, students should now be ready to think enough about geometry to apply light calculations on it. This topic covers the different shading and lighting models. The terminology is usually not that consistent about this. The material uses the term *shading* to denote the choice, where to apply light calculations, and *lighting* to denote, what light calculations would actually be performed.

The introduction mentions that usually light calculations include some values interpolated from the vertex shader to the fragment shader. Definitions include the directional light source and the point light source. In the material only the directional light source is considered, because calculations with a point light source are generally the same.

4.1.6.1 Shading Models

Three possibilities for shading are described and illustrated. Those include:

- Flat shading (per-polygon)
- Gouraud shading (per-vertex)
- Phong shading (per-fragment)

For all of those, the surface normals are shown on the illustrations. It is described that depending on the shading model, the total number of computations varies. That is because historically different shading models have seen a different amount of use.

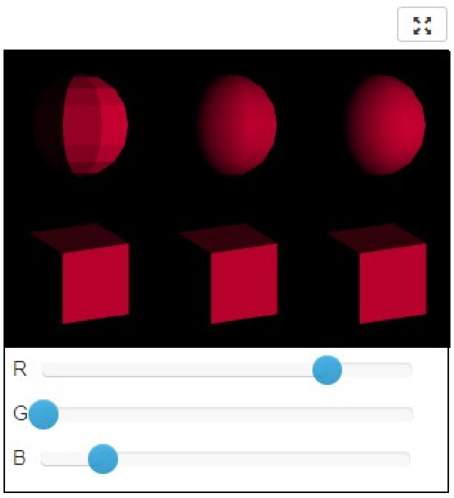


Illustration 22: Example with a sphere and a cube shaded with flat, Gouraud and Phong shadings and diffuse lighting. Sliders are moved so that the objects have a redish color.

There is an example that shows an approximation of a sphere and a cube shaded with all three shading models and the diffuse reflection. It should be visible from the example that for a cube, the flat shading would be the preferred choice, because there is no visible difference. In contrast, if we want to calculate correct lighting for the sphere, then both the flat and Gouraud shading will produce an undesired result. The example has sliders to vary the color of the objects. These should illustrate that understanding this material gives students the ability to create differently colored objects with correct light calculations (Illustration 22).

4.1.6.2 The Lambert Lighting Model

The Lambert lighting model models the diffuse reflection of light from the surface. It also assumes that light reflects to all directions with an equal probability. This is illustrated in the material by describing a photon going inside the material, bouncing around there, and finally getting reflected in a random direction.

The amount of light reflected depends on the amount of light received by a surface unit. This is illustrated by a beam of light, one unit wide, hitting a surface at different angles. On a more perpendicular angle, there will be more light on the unit surface, on a grazing angle, the beam covers a wider area.

The cosine of that angle needs to be calculated. The material describes how the dot product can be used to easily achieve that. This should show students a very useful application of the dot product. The same formula for diffuse light reflection is written out for the red, green and blue channels. It should be more graspable that way, rather than if they were written as a vector equation on the channels that uses also other vectors from our affine space.

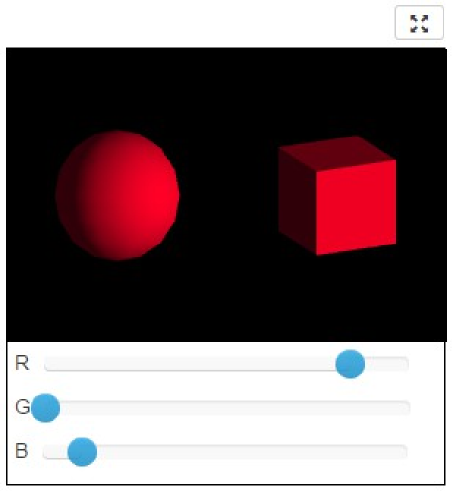


Illustration 23: Example of a cube and a sphere that have the ambient term together with the diffuse term.

4.1.6.3 Ambient Light

Next, the notion of ambient light is introduced. It is explained that usually light will bounce around all over the scene and illuminate every point by some amount. In the previous example there was no ambient term added and students were asked to take a closer look at the parts of the objects, where no light was directly shining upon.

The ambient term is introduced to the lighting calculations and there is an example of a sphere and a cube that now also have the ambient term included (Illustration 23). As before, students can change the color of the material.

4.1.6.4 The Phong Lighting Model

The third part of the simplest lighting models is the specular term. The material explains that very few surfaces actually absorb and then diffusely scatter all the light. Many of the surfaces reflect some portion of the light directly, and only some portion gets diffusely scattered.

The material describes the Phong's specular term that uses a cosine of the angle between the reflected light direction and the viewer direction. The cosine function is plotted and it is shown that we need to raise it to a power in order to get a small highlight.

This specular term together with the diffuse and ambient terms make up the Phong's lighting model that is now presented as a complete formula for all three color channels.

The example shows a specular highlight on a sphere and asks the user to move the slider so that the specular highlight would be small enough. Furthermore, it is explained that the specular reflection should usually have a white color, not the same color as the diffuse or ambient terms (Illustration 24). The example also demonstrates that and asks students to configure the colors so that the specular highlight is visible. Color sliders indicate a full red on the left and full white on the right. When both the specular and diffuse color are full red, then the specular is not visible. The correct configuration would have the diffuse color on red and the specular color on white.

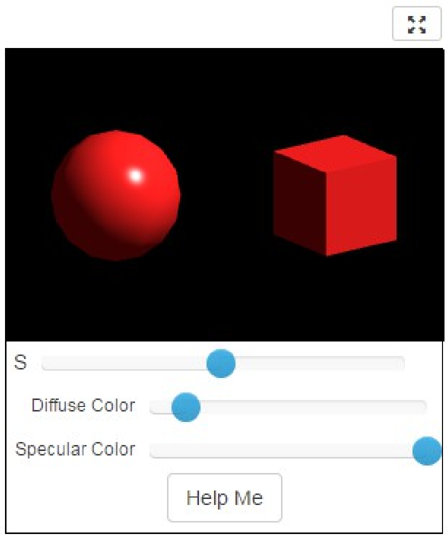


Illustration 24: Example with configurable shininess power S, diffuse and specular colors varying from red to white.

4.1.6.5 The Blinn-Phong Lighting Model

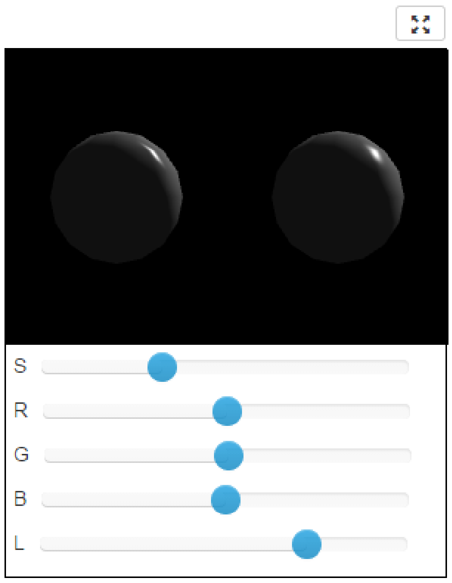


Illustration 25: Example that compares the Blinn-Phong and Phong specular highlights on an approximation of a sphere.

The last material in this topic describes the Blinn-Phong lighting model. The illustration shows the halfway vector between the light and the viewer direction vectors. It is also mentioned and referenced that the Blinn-Phong model produces more realistic specular highlights at grazing angles.

Further description explains that in some cases (orthographic projection, directional light source) the halfway vector can be precalculated, thus increasing the performance of the fragment shader.

The example compares the Phong's specular highlight with the Blinn-Phong specular highlight (Illustration 25). It has a slider for the shininess, the color and also for the directional light's direction. The latter is needed to move the light source to a grazing angle.

4.1.7. Textures and Sampling

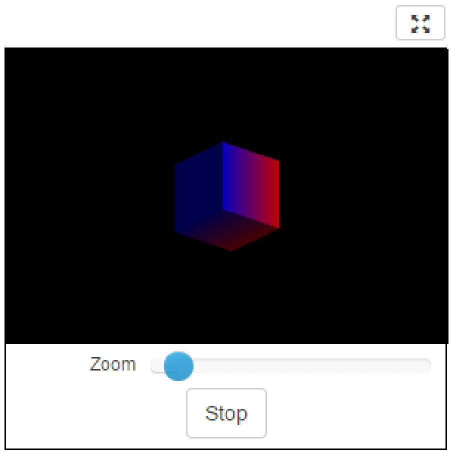


Illustration 26: Example of a cube with a gradient.

After students have learned about the light calculations of a shape with color values specified either per shape, per polygon or per vertex, a natural thing to learn next would be texture mapping that allows to specify more granular patterns inside a polygon.

Introduction to this topic also includes an example with a cube that has a face on one side colored blue and the opposite face colored red (Illustration 26). This creates a gradient on the other faces. This example should show the students, what can be achieved with the material covered this far.

Next, the introduction describes that we could map a mathematical function to a polygon, but similarly we can map an image (that can also be thought as a function) to it.

4.1.7.1 Interpolation

The most important part of texture mapping is how we interpolate the values between the texels. That is why this material is written quite extensively, has a lot of illustrations and is divided to upscaling and downscaling segments.

With upscaling we introduce the nearest neighbour and bilinear interpolations. Upscaling is covered before downscaling, because with upscaling we mostly have only those two options. Of course there are other interpolation techniques, like bicubic, but those are more costly than the prior two. Also interpolating values for upscaling should be more easily graspable because we always have only 4 neighbours from which to interpolate from.

With downscaling the nearest neighbour and bilinear interpolation are also shown first, but after that there is an example, where sampling only the 4 neighbours will produce an undesired result. That gives way for the mipmap material.

4.1.7.2 Mipmap

This material covers the idea that we can pre-downscale the image so that, when sampling points that cover a larger area than the 4 neighbours provide, we can use the already downscaled and averaged mipmaps. The same problematic example from before is shown a solution with mipmaps.

Next the trilinear interpolation is described with an illustration.

Together with a description of anisotropic filtering there is an example of a checkerboard pattern that is viewed at a grazing angle (Illustration 27). The example has buttons to either use no mipmapping, use the bilinear filtering with the nearest mipmap or use the trilinear filtering, where the results from two nearest mipmaps are linearly interpolated. Explanation of that example is also provided and it describes the different artifacts that occur with no mipmapping, the bilinear filtering and without

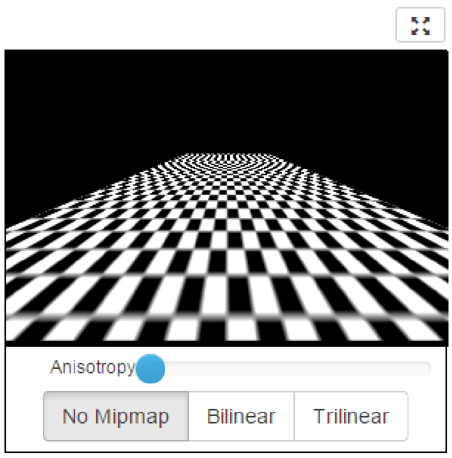


Illustration 27: Checkerboard pattern viewed at a grazing angle.

anisotropic filtering.

This material ends with a brief overview of texture atlases and a possible color bleed that can occur if mipmapping is used together with them. There is a reference there to an article [56] that describes the topic in more detail for the students, who find that interesting.

4.1.7.3 Aliasing

The last material in this topic describes the effect of Moire aliasing, which occurs when we sample a high frequency pattern with a too low sampling rate. This is illustrated by undersampling a sine wave and getting another (alias) sine wave as a result.

The example here is of a function that increases in frequency as it is sampled further away from 0. There is a slider that specifies the step, at which we sample this increasing pattern. When the sampling rate is too small and frequency gets higher, noticeable Moire patterns will appear (Illustration 28).

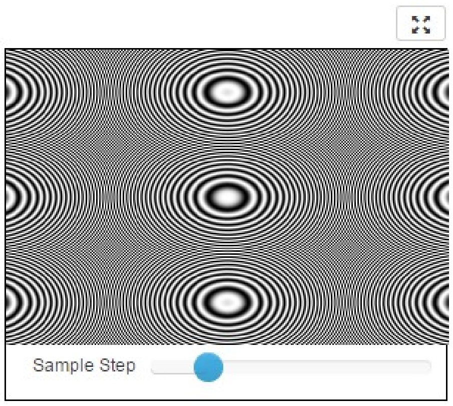


Illustration 28: Example of a function with increasing frequency sampled at a low enough sample rate to produce Moire aliasing.

4.1.8. Blending

This topic was written by Jaanus Jaggo, who also implemented an example occurring here. The main goal of the blending topic is to provide an overview and understanding of the process of rendering different fragments behind each other and mixing together the color values.

4.1.8.1 Depth Buffer

The material points out the perspective projection matrix derived in the Projections material. It should relate to the students that we tried to bring the *z* value along, when doing a projection. The calculations for the *z* value are written out and it is

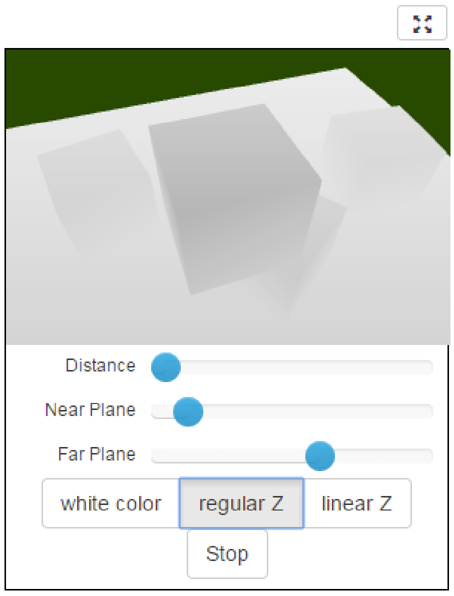


Illustration 29: Example illustrating the depth buffer.

up to the students to see that those are the same calculations that were done with the matrix, and the *w*-divide afterwards.

Then the *z* values are normalized to a range $[0..1]$ and mapped on an integer buffer. The naterial also recapitulates that in the case of perspective projection there is a lot of precision near the camera and it decreases away from the camera. Result of that can be an effect known as *z-fighting*.

Example here shows the depth buffer values that are mapped to a displayable range (Illustration 29). Students have a choice to see the depth buffer values either in a non-linear or linear mapping. There are also sliders to move the camera away from the objects in the scene, and change the near and far planes.

4.1.8.2 Color Blending

After covering the depth buffer, students should have an understanding, how fragments are determined to be in front of each other. This forms the base for the description of the general blending formula presented in this material. That formula is used to demonstrate the conventional alpha blending, premultiplied alpha blending, additive blending and multiplicative blending.

4.2. Basic II

This module consists of a variety of different topics that need a good understanding of the Basic I topics to proceed with. There is no single focus, rather it is a collection of computer graphics related topics that students can find useful in their further studies of the field. The choice of topics was mainly made based on the topics covered previously (Fall, 2013/2014) in this course.

The topics connect with a variety of different other subjects, for example Curves topic needs a base understanding of derivatives from Calculus I (MTMM.00.179) and curve fitting from either Numerical Methods (MTMM.00.005) or Scientific Computing (MTAT.08.010). The Procedural Generation topic includes a bit about formal grammars that are usually taught in the Automata, Languages and Compilers (MTAT.05.085) course. With Ray Tracing and Global Illumination topics we exceed the real-time performance limits of the GPU, which are seen in the Parallel Computing (MTAT.08.020) course. The

Space Partitioning material includes data structures that are often covered in the Advanced Algorithmics (MTAT.03.238) course.

After going through this module, students should be able to investigate different modern computer graphics algorithms themselves with less effort than it would take otherwise.

4.2.1. Modeling and File Formats

This topic was written by Timo Kallaste. The reason for it being the first topic in this module is that the students at this point have modeled their geometry using simple geometric primitives. It serves as a fitting continuation for them to now be able to import models made in a specific modeling software and see that the transformations and lighting calculations apply to those models as-well.

From the organizational perspective of the course, the practice session is the same for the Basic II and Game Engines module students. The latter will start to then delve further into different modeling techniques and game engine possibilities.

4.2.1.1 OBJ

This material describes the OBJ file format that is often used to model static geometry. In the material it is emphasized that the reason lies in the fact that OBJ does not keep the object's hierarchy. This should relate to the scene graph topic for the students.

4.2.1.2 FBX and Collada

The material here describes the more complex file formats that do preserve the object's hierarchy and also allows other data, like animations, to be stored and transferred.

4.2.2. Environment Mapping

The Environment Mapping topic first introduces the notions of the sky box and the sky dome. The description is again written in a problem-oriented fashion, where the question is, how to fill the background of our scene with something meaningful. Mapping a texture to the sky box or a sky dome is already considered environment mapping. The further material shows how to create reflective objects by mapping the existing environment onto them.

4.2.2.1 Cube Map

This material describes first our goal to create a reflective object in the scene. In order to further illustrate that, the derivation of reflecting an incident vector from a point is given.

The material then describes that sampling the reflection directly from the sky box is an approximation. The images on the sky box are considered to be infinitely far away and thus the location, where we reflect the incident vector, can be considered to always be the center of the scene. In order to solve that, a cube camera (6 perspective cameras) can be situated in the object's center and the scene can be rendered from that every frame. Those will create another cube map, from which we can sample to make the object reflective. The material has illustrations of that and also mentions that this has drawbacks for self-reflections.

The example accompanying the material is a scene, where a cube map of Tallinn's Town Hall Square is mapped to the sky box, there is a reflective sphere and the camera moves around that sphere (Illustration 30). The images of the cube map are taken by E. Persson [57]. Student has a possibility to try out a cube or a tube shape instead of the sphere. There is also an effect that creates a number of floating colored spheres in the scene. This should show that in order to get also the reflections of those colored spheres, we do have to render the scene with the cube camera. Otherwise we do not know, how they currently look from the reflective sphere's perspective.

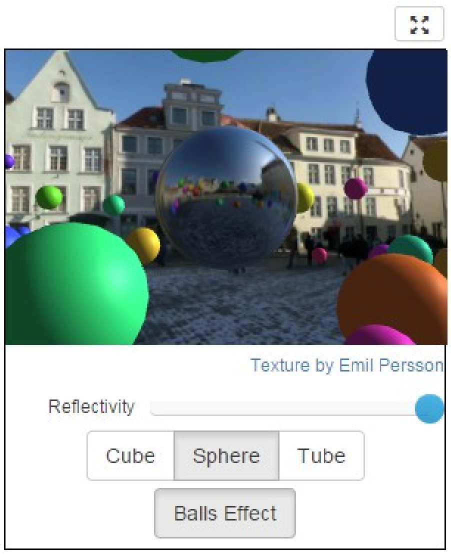


Illustration 30: Example of a cube map, where a number of colorful spheres are floating around. They are also seen from the reflection.

4.2.2.2 Sphere Map

There is also a brief material on the sphere map that describes the technique and why it was historically used first. The comparison with a cube map is provided.

The example enables the student to rotate the camera via a slider and turn on and off the sphere map regeneration. If the regeneration is turned off, and the camera is rotated 180°, then the sphere map has visible loss of detail and a blind spot exactly behind the sphere, opposite of where the previous sphere map was rendered from (Illustration 31). The example also allows the student to actually see the rendered and mapped sphere map.

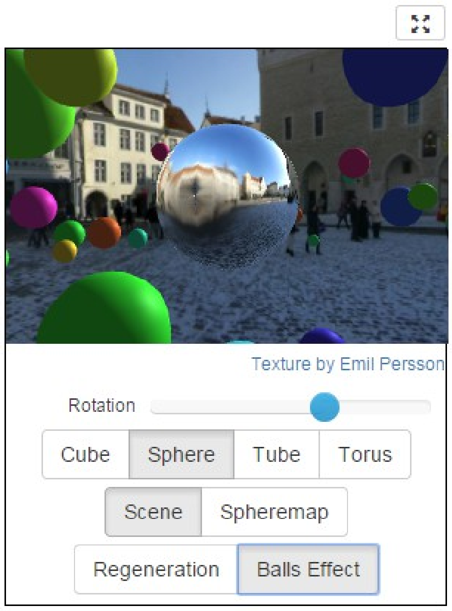


Illustration 31: Sphere map example, where the sphere map regeneration is turned off and the camera is rotated so that the loss of detail and a blind spot is visible.

4.2.3. Curves

So far students had only seen rough approximations of curved surfaces. In reality, many things are modeled using curves and curved surfaces. Also, with the advent of tessellation / geometry shaders, the construction and properties of curves have become important topics.

Introduction starts by describing the tessellation, again in a problem-oriented approach. Then it offers a solution in the way of an interpolating cubic polynomial curve, together with the derivation of it. During that, the algorithm for constructing a curve, given a number of constraints and the parameter vector, is also presented.

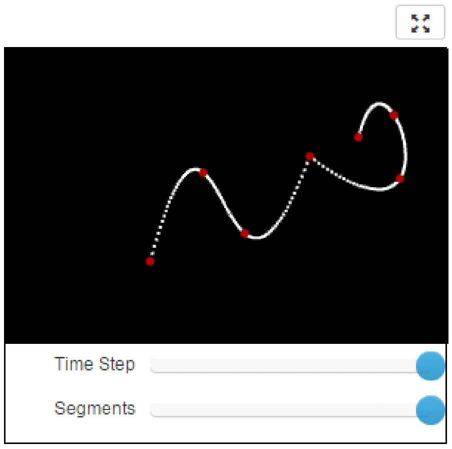


Illustration 32: Example of two manipulatable cubic interpolating polynomial curves. The entire spline is not C^1-smooth.

After constructing the curve, a notion of a spline is introduced and the focus shifted to constructing many curves that fit together in some smooth way. This is also, where the formal definition of smoothness of a curve (or spline) is given.

The example has the previously constructed cubic interpolating polynomial curve. Student can drag the red control points around to change the shape of the curve. There is also a slider that adds another 3 control points and another interpolating cubic curve (the next segment of a spline) to it. It is visible that with this construction, generally the spline will not be C^1-smooth.

4.2.3.1 Hermite and Cardinal Curves

Students are reminded that for the entire spline to be C^1-smooth, the derivatives at the knots need to match. Another set of constraints are constructed to satisfy that condition and a spline is found with the same method as before. This shows the students that modifying the constraints will create another curve, while the actual construction process stays the same.

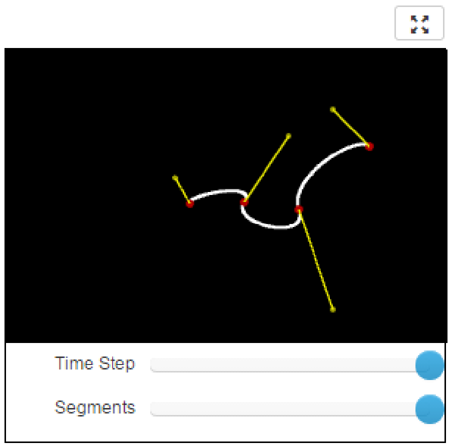


Illustration 33: Hermite spline example. Students can interact with the control points (red) and the derivative parameters (yellow) directly, by dragging the corresponding points.

In the Hermite curves part, the constraints on the derivative are taken by specifying two of the parameters to indicate the derivatives. This means that the derivatives can be modified, by changing the parameters.

The example shows such a spline, where students can modify the parameters for the control points and the parameters for the derivatives directly (Illustration 33). As seen from the example, and also described in the text, constructing such a spline is a bit unintuitive.

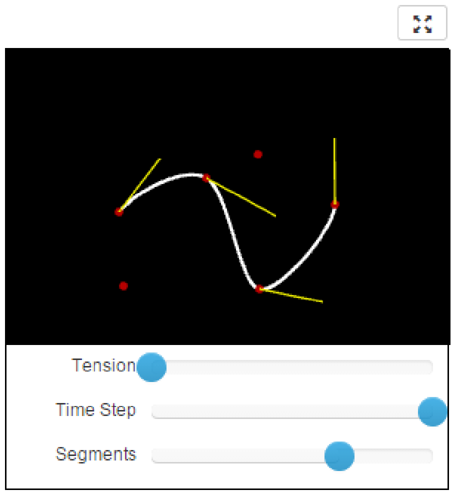


Illustration 34: Cardinal spline example. Yellow lines indicate the derivatives, curve passes all but 2 of the control points.

That problem is tackled with Cardinal curves, where the derivatives are calculated from the previous and the next control points. This shows the students that it is possible to use control points, through which the curve generally interpolates, to find and constrain the derivatives automatically.

A special case of Cardinal curves, the Catmull-Rom curve, is also described.

The example includes a Cardinal spline that has the found derivatives shown on it, but modifying them means modifying the interpolated control points. This should be more intuitive to control the shape of the of the spline and still keeping it C^1 -smooth. There is also the tension slider, where $tension=0$ creates the Catmull-Rom spline.

4.2.3.2 Bezier Curve

This material starts with the geometrical construction of the cubic Bezier curve. It shows to the students that it is also valid to come up with a constructive algorithm for something. That construction can lead to the same conclusion as the mathematical derivation would. The material does exactly that. The de Casteljau's algorithm is presented together with an analysis that derives the basis and constraint matrices of the Bezier curve.

Important properties of the Bezier curve, affine invariance and boundedness by the convex hull, are emphasized.

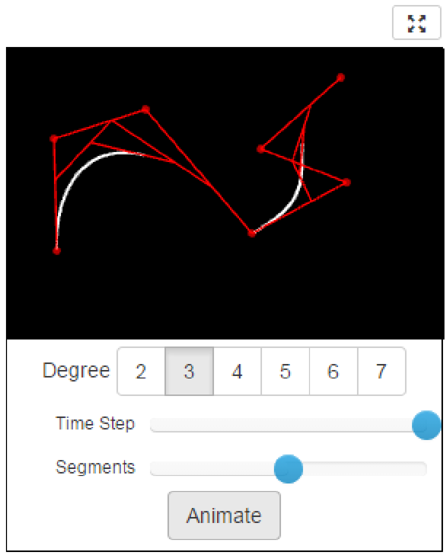


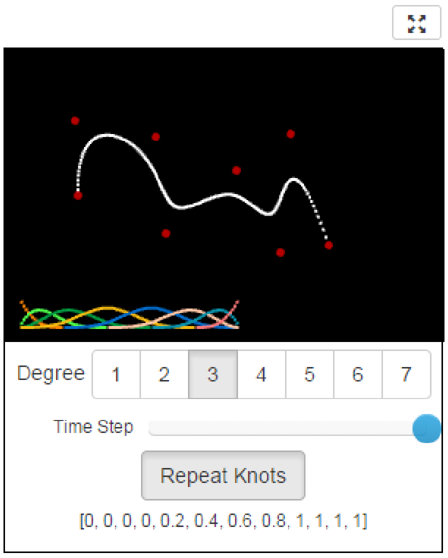
Illustration 35: Bezier spline example that is running an animation of de Casteljau's algorithm for a cubic spline.

The example presents a Bezier spline, where the degree of the curves can be changed. It is explained that in the example each segment is maximally smooth, but the entire spline is only C^0 . Brief description, how to construct smoother splines, is given and there are extra references ([58] and [59]) for further reading. One of the tasks also includes constructing a smoother cubic Bezier spline. Example also has an animate button that shows the steps of the de Casteljau's algorithm.

4.2.3.3 B-Spline Curve

Here students are given a closer look at the blending functions, derived and used before. The material explains that if we create higher degree blending functions recursively, a smoother approximating curve can be constructed. That is the basis of constructing the B-Spline. It is explained that if we take the blending functions to be of high enough degree, then we can create a smooth enough curve. It is also explained that if we repeat the knots at the endpoints, when constructing such a curve, we can make it interpolate the endpoints, without losing smoothness. A more detailed description of this construction is referenced in [60].

The example illustrates such a technique and shows manipulatable B-Spline curves up to the 7th degree. The knot vector is also shown and a button can be used to toggle the repetition of the end knots. Furthermore, the actual blending functions are drawn in the example, on the bottom left part (Illustration 36).

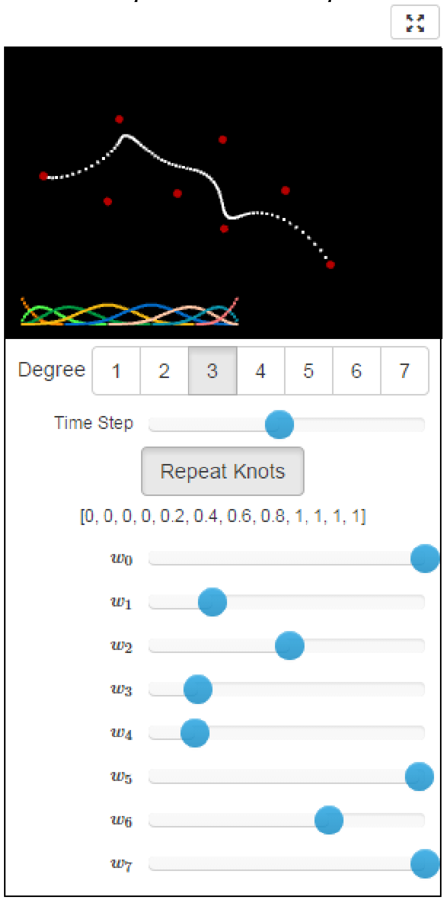


[0, 0, 0, 0, 0.2, 0.4, 0.6, 0.8, 1, 1, 1, 1]

Illustration 36: B-Spline curve example, where a 3rd degree functions are used for blending. The curve is C^2 smooth. The knots are repeated at the ends, the curve interpolates the endpoints.

4.2.3.4 NURBS

The last material in the curves topic describes the non-uniform rational B-spline curves. The material on the B-Spline curves and Bezier curves should be enough to show students that different weights can be assigned to the control points, thus creating the NURBS curve. It is mentioned that the NURBS curves find a lot of use in modeling curved geometry. This should give the students a background to



[0, 0, 0, 0, 0.2, 0.4, 0.6, 0.8, 1, 1, 1, 1]

Illustration 37: NURBS curve example. Different weights can be changed for the control points.

understand the NURBS, when they come into contact with it in their further studies in the computer graphics field. In the end of this material, there is a reference [61] to another material that explains the same concepts from a bit different perspective.

The example is similar to the B-Spline curve example, but with different sliders to change the weights of the control points (Illustration 37).

4.2.3.5 Procedural Generation

Procedural generation is a vast topic and there are many techniques that can be implemented with a varying level of granularity. This is described in the introduction, and here we just cover briefly the basic generation from noise, the Lindenmayer-systems and the particle systems, including the Boids algorithm. Procedural generation falls under the computer graphics, because here we focus mostly on the visual results of the procedural generation. Students have so far leaned about creating geometry from simple geometric primitives, modeling it in a specific modeling software, but generation of complex geometry or textures procedurally is also a valid and useful option.

4.2.3.6 Noise

This material describes first the basics of random values, and then proceeds to describe value noise, which can be though as a sum of interpolated random values of different frequency. Illustrations of that are shown and different uses of such procedurally generated image are mentioned. The source of that algorithm [51] is referenced. This is proceeded by the description of Perlin noise and Ken Perlin's original algorithm [62] is referenced.

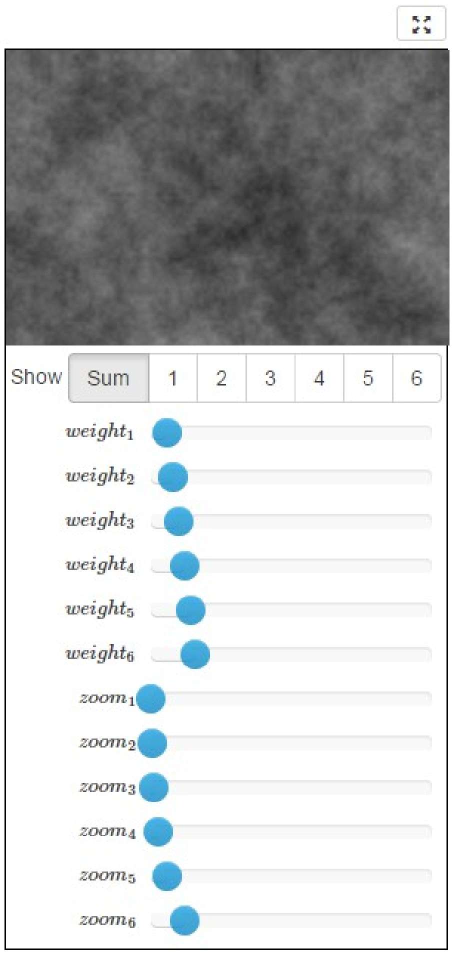


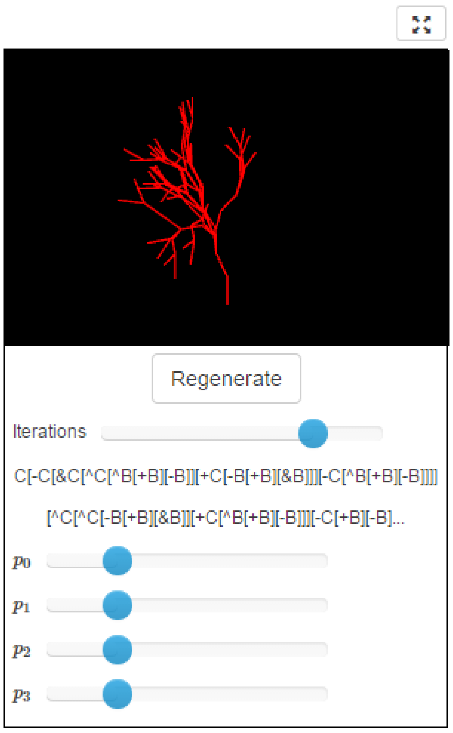
Illustration 38: Example of value noise, which is a weighed sum of differently sampled noise texture.

The example of both value (Illustration 38) and Perlin noise enable students to zoom into the random texture with a varying level of zoom. It shows the sum of the different levels, each level separately and also allows to specify, with which weight each of the levels are considered in the sum.

4.2.3.7 L-Systems

Context free grammars are described and a connection is mentioned between formal grammars and their use of analyzing program code. The notion of self-similarity is introduced. Analogy between program code and procedural generation is stated. From that, a derivation of the stochastic systems is constructed and this results in context free L-systems. Reference to Lindenmayer's original article [63] is given and a simple example is also shown.

The example is based on that example and allows the student to change the probabilities of rules, the number of iterations, and to regenerate different tree-like structures. The latter are drawn with red lines and rotated around. It is mentioned that if the probability of one of the rules would become 1, we would no longer have a meaningful stochastic system.



```
C[-C[&C[^C[^B[+B][-B]]][+C[-B[+B][&B]]]][-C[^B[+B][-B]]]]
[^C[^C[-B[+B][&B]]][+C[^B[+B][-B]]]][-C[+B][-B]...
```

Illustration 39: Example of a L-System. The currently generated word is shown and sliders allow to change the probabilities of rules and the number of iterations.

4.2.3.8 Particle Systems

The last material in the Procedural Generation topic explains the notion of particle systems. These are quite extensively used in computer graphics to create a variety of effects. First a very simple idea of an emitter and particles with a random lifetime is introduced.

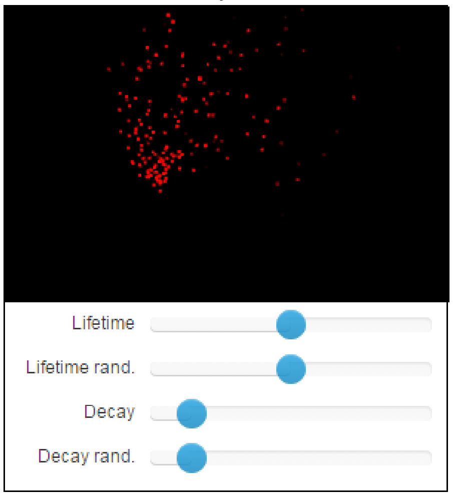


Illustration 40: Particle system example. The lifetime, decay and corresponding random ranges can be modified. Emitter moves around in a circle.

An example of that includes an emitter that moves around in a circle and emits particles from a pool. Each particle has a random direction and a lifetime. Students can modify the lifetime and a random range, from which a value will get added to the total lifetime. Similarly they can modify the *decay* value, from which the particles will start to get more transparent (Illustration 40).

After this it is described that particles could also have rules, according to which they move around. Also there could be a fixed number of particles that will always exist. This creates the basis for the Boids algorithm, for which the rules of cohesion, separation and alignment are described. Some further modifications and additions to the algorithm are referenced ([64] and [52]).

The example shows a flock of particles following the rules of cohesion, separation and alignment (Illustration 41). The student can modify the importance of cohesion, by changing the coefficient, the mass center is taken into account with. *Repulsion* and *Repulsion th.* (threshold) sliders configure the separation rule, and speed center modifies the alignment rule. There is also a slider that implements an additional rule of the particles following a certain target. That target will move around in a circle. The slider to control that rule is called *Trajectory*. In this example, all the particles take into account all others *ie* have a global neighbourhood. There are examples, where only a certain radius is taken into account.

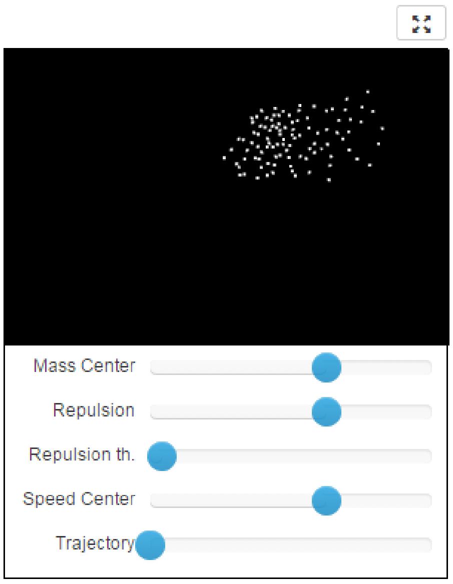


Illustration 41: Example of the Boids algorithm, with sliders to modify each of the 3 base rules. Also a slider to make the particles follow a moving target.

4.2.4. Ray Tracing, Space Partitioning, BVH

The Ray Tracing topic shows the students another way to render a scene, and also prepares them for a global illumination technique called path tracing and detecting if an object is in shadow. The introduction starts with the idea of rendering by casting rays into the scene for each pixel, and draws a comparison that so far the rendering has been the other way around. Ray-triangle intersection detecion is briefly mentioned and it is noted that this can become slow if the geometry is not kept in a specific data structure.

4.2.4.1 Ray Casting

Material starts with a formal parametric definition of a ray and proceeds to derive the Möller-Trumbore ray-triangle intersection algorithm [65]. The points in the triangle are represented in Barycentric coordinates, with 2 degrees of freedom. These degrees are represented as *u* and *v*. The derivation continues by equating the parametric representation of a triangle and a ray. Next a system of linear equations is solved with Cramer's rule and a scalar triple product is used to calculate the solutions using coordinates. Different tests for the final solution are described. The original article that describes this is also referenced.

After this it is mentioned that we can use ray casting (casting a single ray and finding the closest triangle it intersects) in a number of different situations. Especially, if we just want to know, what other object is the closest in a certain direction from one object.

4.2.4.2 Ray Tracing

Ray tracing is explained in a problem-oriented approach, where the problem is to model a reflective surface. The idea of tracing a ray to the reflection, to find the incident light, is described. Refractions inside a semi-transparent material are also mentioned. The Material deliberately ignores the concept of shadow rays at this point.

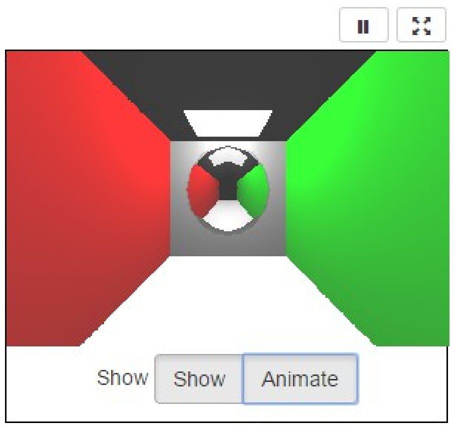


Illustration 42: Example of a ray traced scene with a reflective sphere.

The example consists of a ray traced scene, where there is a moving reflective sphere inside. Because this is a costly algorithm, a pause button is added to the example and initially the example starts in paused mode. There are buttons to control the existence and movement of the reflective sphere (Illustration 42).

4.2.4.3 Space Partitioning

This material briefly mentions space partitioning data structures: octree, K-D tree, binary space partitioning. They are described by a problem-oriented approach, where we have 15 rays testing intersections with a 20-sided icosahedron. Each method has an illustration and the total number of ray-triangle intersection tests is estimated. Material references a nearest neighbour search demo [66] that further illustrates the ideas behind the octree and K-D tree.

4.2.4.4 Bounding Volume Hierarchy

Besides the space partitioning methods described in the previous material, in computer graphics, we usually want to also create a bounding volume hierarchy. Benefits of this are mentioned in this material.

4.2.5. Global Illumination

This topic starts by describing the local illumination done in the standard graphics pipeline way of rendering and compares it to the actual global illumination. Specifically, the ambient term is focused on. The material also mentions the Cornell Box scene. The rendering equation is introduced, briefly described and illustrated.

4.2.5.1 Path Tracing

The algorithm of path tracing with direct lighting [67], is described and illustrated in this material. Different descriptions include the cases, when the random ray: hits the light source, hits another object, hits nothing. The recursive formula for this is shown and, as before, the shadow rays are ignored.

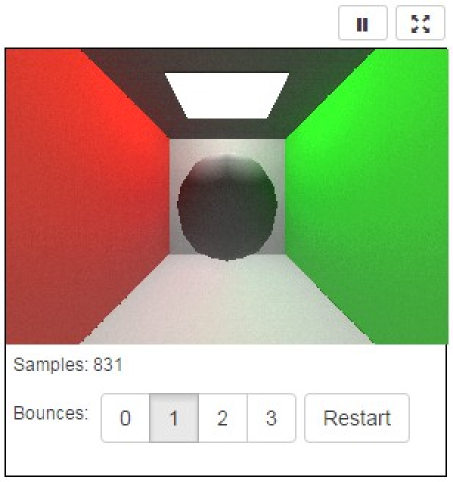


Illustration 43: Path trace example that has rendered and averaged together 831 samples. The number of bounces for the rays is 1. Indirect illumination from the walls is visible on the sphere, the floor, the back wall and the ceiling.

The example has a path traced scene, where there is an approximated sphere in the middle of a room. Students should see that with bounces higher than 0, the indirect illumination from the differently colored walls, is shown on the floor, the back wall, the ceiling and the sphere. Because this is again a computationally heavy algorithm, the example includes a pause button and starts paused. The number of samples averaged together is shown (Illustration 43). It is visible that if the number of samples is low, the render is noisy.

4.2.5.2 Photon Mapping

This material mentions that there are different global illumination algorithms, like radiosity and photon mapping. Then it describes and illustrates the main idea behind photon mapping. Students should see that there are many quite different approaches for realistic scene illumination.

4.2.6. Shadows

The Shadows topic was left last, because it is more clear if the students have already covered global illumination (especially path tracing) before this. Reason being that with global illumination we can easily get the correct shadows, thus it provides a good comparison with the usual shadow algorithms.

The introduction describes the essence of shadows and illustrates them for an area, a point and a directional light source. Definitions for umbra, penumbra and antumbra are given. A material that includes different techniques for soft shadows in the case of a directional light source [68], is referenced.

4.2.6.1 Global Illumination Shadows

This is where the idea of shadow rays is explained. Students, having previously just seen path tracing, should be able to connect the shadow rays with that easily. It is illustrated that doing shadow rays for a point light source, still does not give us soft shadows, so an area light and random shadow rays to it are used instead.

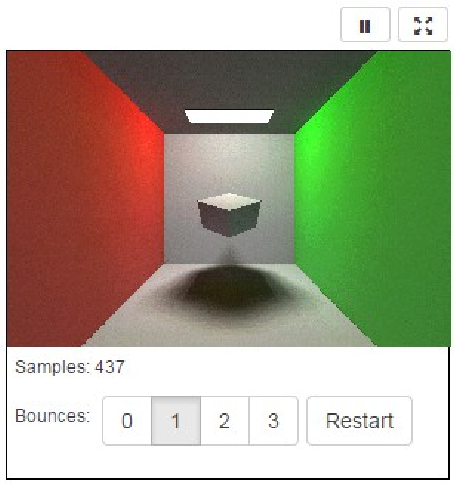


Illustration 44: Example of a path tracer with shadows. The umbra and penumbra of the cube are visible on the floor.

The example includes a modified path tracing example, where there are shadow rays shot at the area light source for each hit point. From this example, students should be able to see the penumbra and umbra parts of the shadow (Illustration 44).

4.2.6.2 Shadow Mapping

Here the current popularity of shadow mapping is mentioned. Then the material proceeds with a problem-oriented approach to derive the shadow mapping algorithm. Illustrations and descriptions are made for the point and directional light sources. Problems with the anisotropic sampling of the shadow map and the need to specify a big enough view volume for the shadow camera, are mentioned. The first problem can be solved with trapezoidal shadow maps and their description [69] is referenced in the material.

The example is of a scene with two rotating green spheres, rotated back wall and a yellow box. Each of those casts a shadow from the point light source located in the center of the scene (Illustration 45). Controls allow students to see the different shadow maps rendered with a cube camera from the light source. The depth values are converted into a visible range of monochrome colors. There are also controls to change the resolution of the shadow maps. Another button allows to take more samples from the maps and seeing if the current fragment would be at the edge of a shadow, in which case more of the light is considered. Students should be able to see the aliases on the edges of shadows and that they get worse, if the shadow map resolutions are lower.

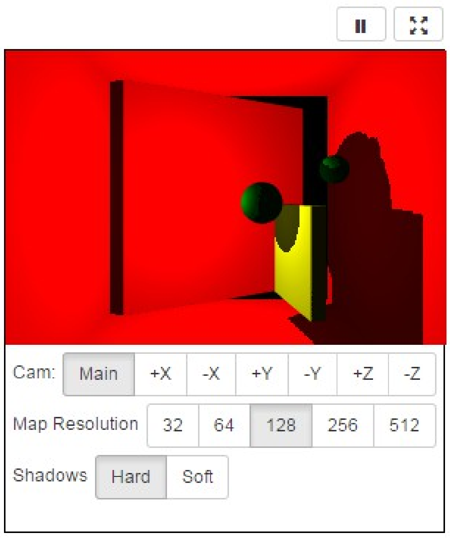


Illustration 45: Shadow mapping example. The aliasing on the edges of shadows is visible. Currently a 128×128 sized shadow maps are rendered.

4.2.6.3 Shadow Volume

The last material in the Shadows topic describes the shadow volume technique that is also called stencil shadows. Material begins with describing, how to find the silhouette edges of a polygon from the light source. Then it proceeds with constructing the volume by duplicating the vertices on the silhouette edges, making faces between them, and extruding the duplicates with the dark cap away from the light. After this, students are asked to think, how far should the dark cap be extruded.

The answer to this question is *to infinity*, after which, the projection matrix is enhanced to have the far plane in infinity. Then it is explained how points in infinity can still be seen or visualized in the projection. Students should be able to connect this material with the perspective projection matrix presented in the Projections material.

Finally, the step-by-step shadow volume algorithm is presented.

The example has the same scene, as in the Shadow Mapping material, but this time it implements the shadow volume algorithm. Different buttons allow to render the shadow volumes (which can have configurable back or front face culling), the shadows rendered with a white color, or the scene rendered normally (Illustration 46). Students should be able to grasp, how the shadow volumes actually look like, when rendering the shadow volumes only. They should also see that the edges of shadows here do not produce aliasing that occurred in the shadow mapping example.

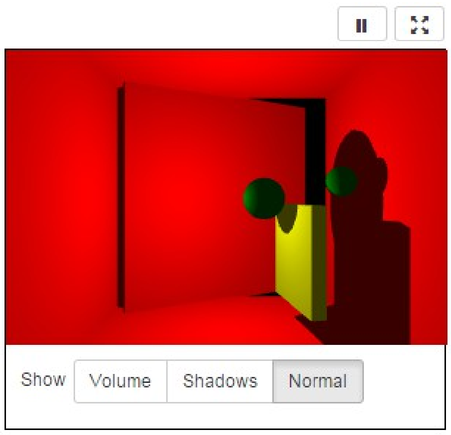


Illustration 46: Shadow volume example. Normal rendering is shown. The shadows do not have aliasing on the edges, as was the case with shadow mapping.

4.3. Game Engines

The entirety of this module's material was written by Ats Kurvet and Timo Kallaste. Details of that material are not part of this thesis. The aim of the module is to provide students with a higher level approach to computer graphics in the form of using the knowledge from the course in a modeling software Blender and a game engine Unreal Engine 4. With that in mind, the material was constructed to support different key aspects of the two pieces of software. Often the material covers similar aspects that are taught in the lectures, for example Rigging and Animation topic is connected with the Curves topic, because curves are used to specify how different animations are performed.

Overall freedom was given to Ats Kurvet and Timo Kallaste to construct the material in a way that would benefit the students most from their perspective.

5. The Tasks

Initially the idea was to have tasks for each material, but when creating the tasks, it seemed preferable to have tasks per topic. This creates more freedom to have different tasks that are related to the topic, but can cover material that is not explicitly written under the materials. The granularity of the material also would have not been suitable for the tasks.

In this thesis the focus is, again, on the tasks that are in Basic I and Basic II modules. The creation of tasks in the Game Engines module was assigned to Ats Kurvet and Timo Kallaste, with a freedom to create them as they saw fit.

The tasks start out with a description of the technique that is part of the solution. Often the ideas under the material were repeated, but with a more practical perspective. For example, in the Cube Chopper task, the idea of front- and back-facing triangles was important and an illustration of a cube with indexed vertices was shown. Similarly, the descriptions had references to external sources for additional reading.

After this, a goal of the task was stated. This included solid statements, what should be done in the task. Together with the statements, there were also screenshots that showed, how the final result should look like. This is important from the student's perspective, because that way they can look at the screenshot and see right away, if their result resembles it or not.

Finally, additional guidelines for JavaScript and C++ were given, together with the links for the base code for both of them. Base code usually included a program that had missing pieces of the parts required for the technique the task was based on. This allowed the students to have some code that already worked (or would work with minor modifications) and they could focus on the actual problem attributed to the task. The base code in JavaScript was made by me (with the exception of the task Soft Particle Chopper, which was made by Jaanus Jaggo), and most of the base code in C++ was made by Margus Luik. The C++ base codes, were constructed as Code::Blocks [70] projects, which was the choice of software in the course.

JavaScript solutions focused on Three.js, while still having the students first implement some low-level algorithms, before using the corresponding classes from the library. The goal of those tasks was to not only teach students computer graphics related algorithms, but have them see that modern browsers can also easily render 3D graphics. The more computation-heavy algorithms would become slow however, because of the bottleneck of JavaScript that is interpreted by the browser.

C++ tasks were made with OpenGL 4+, using different libraries like GLM [71] and GLFW [72]. This approach was meant to show students also a lower level approach to computer graphics. Because students vary in their previous skills and further goals, it is important to provide them with a couple of alternative ways, from which they can pick the one that suits them the most.

5.1. Basic I

5.1.1. Computer Graphics

The first week started with 5 tasks that each had the same problem, but used different environments. The idea was to give students some initial experience with both JavaScript and C++ technologies. All the tasks consisted of finishing a function that draws an equilateral triangle at a certain position. This problem had the students first thinking about the geometry of an equilateral triangle, and fixing the one degree of freedom, they had, in order to specify one. Then it showed an example of a software design question: if one has to complete a function to draw an equilateral triangle at a given position, what would be the most logical approach. There was even some discussion about that, if it would be more logical to fix the inner radius, cirumradii or the side length.

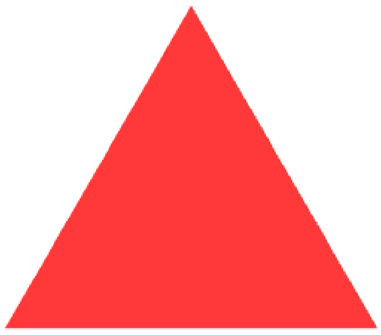


Illustration 47: Equilateral triangle required to be drawn in one of the tasks in the first week.

5.1.1.1 Hello Canvas

The equilateral triangle was required to be drawn, using HTML canvas drawing functions. This required students to browse the documentation and find out, what functions to use and how to use them in order to draw a predetermined path.

5.1.1.2 Hello WebGL

The task introduced WebGL as a drawing context. This introduced the notion of shaders, vertex arrays, and the normalized device coordinates. Students also had to consider the aspect ratio of the viewport.

5.1.1.3 Hello Three.js

In contrast to WebGL, the Three.js graphics library wrapped many of the previous low-level things. Students had to consider the THREE.Vector3 and THREE.Face3 classes, in order to draw the triangle. Emphasis was put on the fact that the triangle face had to point towards the screen, ie vertices be in the counter-clockwise order.

5.1.1.4 Hello Allegro

This introduced the 2D graphics programming library Allegro for the students. The task illustrated an infinite event loop and Allegro functions for creating windows and drawing.

5.1.1.5 Hello OpenGL

The task showed the use of GLFW for creating windows and used older OpenGL 3.0, to give students the background information, how graphics were done before the newer OpenGL versions.

5.1.2. Introduction to Geometry

This week consisted of 2D graphics drawing, rasterization and Barycentric coordinates. We used 2D graphics, because 3D was not required yet, and we wanted to give students the experience of dealing with 2D. JavaScript tasks used the canvas drawing functions and C++ tasks used Allegro. In JavaScript, specific helper classes were constructed, to ease the manipulation of points and colors.

5.1.2.1 Bresenham Line

This included the standard Bresenham line rasterization algorithm. The base-code was structured to have students implement all the choices for a line (steep descending, steep ascending, non-steep descending, non-steep ascending) in separate branches. The idea was to show students that in some cases there are algorithms that need to be extremely fast and thus the code can sometimes be less compact. This also introduced the idea of rasterization for the students.

Some students did try out different implementations (including compact ones) in this task. The performance of those solutions was compared against the recommended implementation using jsPerf [73].

The base code included lines that were drawn with ordinary canvas or Allegro line drawing functions and had the students to draw a similar picture with the Bresenham algorithm (Illustration 48).

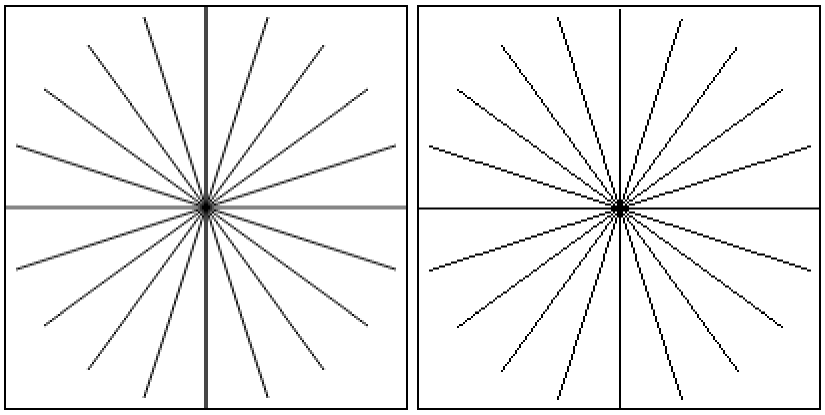


Illustration 48: Example output shown in the task description. Left image has lines drawn with canvas drawing functions and the right was the expected output from the Bresenham line algorithm.

5.1.2.2 Bresenham Triangle

The task introduced students to the idea of Barycentric coordinates of a triangle. This was an important idea to cover, before moving on to shader interpolated values in 3D graphics.

Students were asked to use a similar idea, as in the Bresenham's line, to rasterize the lines between the vertices of a triangle. This should be done in a way, where at each iteration, a vertical (or horizontal in C++) line could be constructed between two rasters. That line was further rasterized. When rasterizing, the Barycentric coordinates were required to be calculated and colors assigned to each vertex interpolated to a corresponding pixel.

The JavaScript base code included a rotation animation of a triangle, and the notion of an event loop in JavaScript was introduced. It also included the outline of the triangle drawn with canvas drawing functions for reference to the students Illustration 49. C++ code included a triangle drawn in different positions.

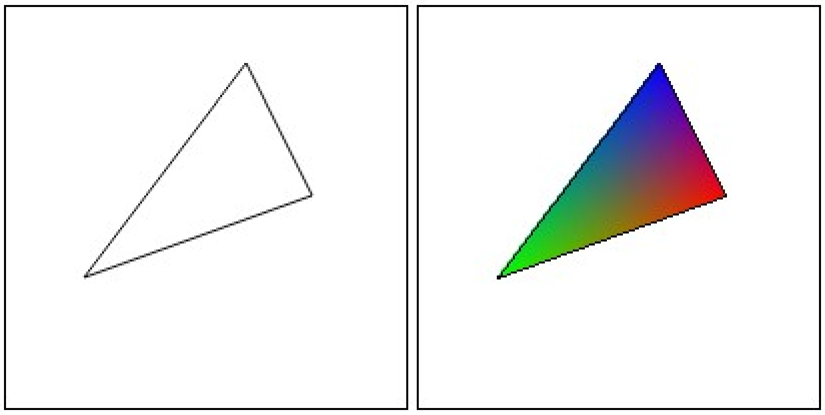


Illustration 49: Still image of a triangle outline drawn with canvas drawing functions for reference on the left and the expected outcome from the task solution on the right.

5.1.2.3 Wu Line*

This task was requested by one of the students, and thus was created as a bonus task. No base code was given, but task description included the instructions for rasterizing a line with the Wu line algorithm and a reference, from where a pseudo-code could be found.

5.1.3. Geometry and Transformations I

This week marked the start of 3D graphics and also introduced the Chopper scene. This is a scene that is prominent in a number of subsequent tasks. The idea was that students will then be familiar with a one scene where they can try and implement different algorithms. The Chopper scene includes a hangar with 3 walls, ceiling and floor modeled as rectangles. In the center there is a helicopter that consists of a body and two rotating blades.

The topics Geometry and Transformation I and the Geometry and Transformation II were covered in one week.

5.1.3.1 Cube Chopper

Students were asked to manually create a cube by specifying vertices and faces. That cube would be basis of the body and the blades for the chopper. The cubes were to be assigned in an hierarchy and different transformations applied to them, to make the result resemble a chopper. Illustrations were provided to help the students specify the correct order of vertices to make the faces face outwards. The chopper was colored with uniform colors (Illustration 50).

For the JavaScript task, students were told to avoid the THREE.BoxGeometry in this task, but consoled that they could use that in the subsequent tasks. In C++, OpenGL 4+ was introduced together with the GLM library to create transformation matrices. A matrix stack was used to specify the hierarchy. The function to draw the hangar walls, gave an example of its use.

The blades were requested to have a rotation animation. This had the students think about the time in which one iteration of the drawing loop takes place, and how to make the rotation occur with a given speed, independent of the performance of a computer.

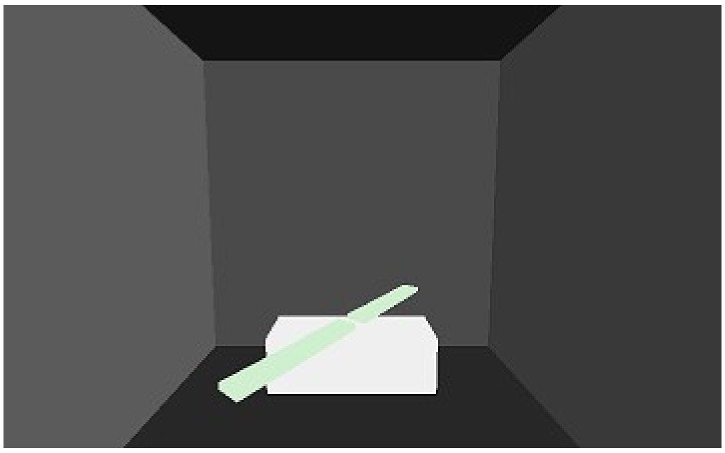


Illustration 50: Still of the chopper to be created in the Cube Chopper task.

5.1.3.2 Flying Chopper

This task had the students try to implement some additional things that they would find most interesting. Those included:

- Move the chopper by sampling some function.
- Learn how to fetch user input and transform the chopper according to it.
- Vary the speed of the blades rotation by the chopper's own movement.
- Manipulate the camera.
- Collision detection.

5.1.4. Geometry and Transformations II

This topic was covered together with the previous one.

5.1.4.1 Shader Chopper

This task had the students further research the shaders. Specifically the task consisted of sending assigned colors for the vertices as attributes to the shaders, and interpolating the fragment color based on the vertices. This should remind the students the Bresenham Triangle task, where colors were also interpolated.

The task also required the blades to both have the darker ends in the middle and lighter ends pointing outwards (Illustration 51). Solutions to this included rotating one of the blade cubes 180°, or specifying another parameter in the shader's construction function that switched the colors. The solution where one of the blades was scaled with a negative coefficient in one axis, was wrong. This was explained to the students, who submitted that solution, together with an illustration, why this was wrong.

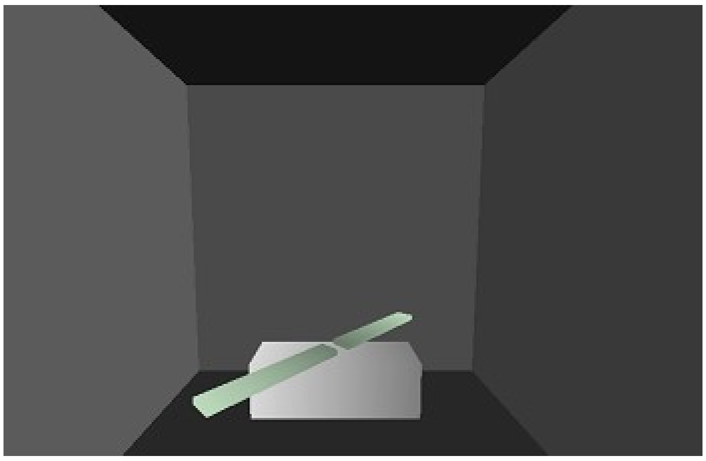


Illustration 51: Still image showing the boxes of the chopper being colored with gradients.

5.1.5. Frames of Reference and Projection

This week lowered the pace of tasks a bit, for the students to think more about the material covered so far. There was only one task that included the manipulation of different cameras to get a feel of them.

5.1.5.1 Projected Chopper

The task had the students change the field-of-view (FOV) parameter of the perspective camera based on the user input. The task also posed a question, what happens if the FOV is below 0°, or over 180°.

Secondly, an orthographic camera was supposed to be constructed and changed to show the orthographic projection of the chopper from the top, front and side (Illustration 52). Configuring the orthographic camera had the students also think about the world space, in order to assign the correct up-vector for the camera. That alternation was switched every 3 seconds in the JavaScript code, or based on the user input in the C++ code. Idea was to show the students the orthographic projection and remind them how it looks (because this is often covered in engineering drawing classes in high-schools, especially the front, top and side views of objects).

The result of the orthographic projection was to be shown in the bottom right corner. To achieve this, students had to configure different view ports and enable scissor testing.

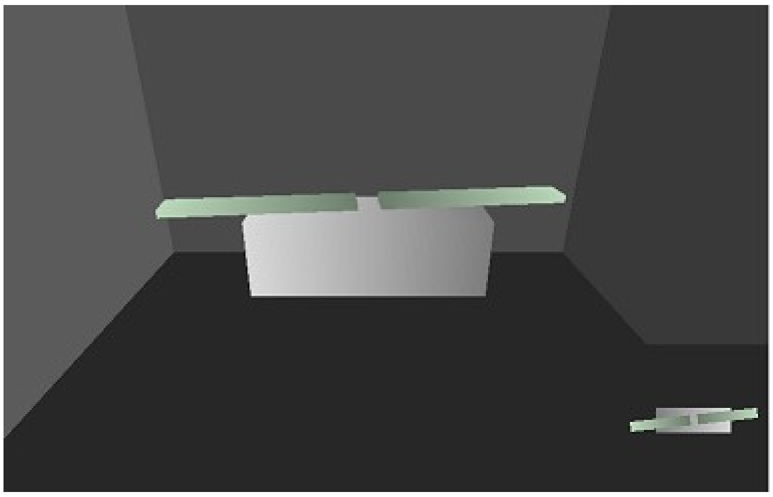


Illustration 52: Perspective camera has the FOV changed and an orthographic projection of the chopper from the top is shown in the bottom right corner.

5.1.6. Shading and Lighting

Here the tasks consisted of implementing two different shading and lighting models and also research and implement gamma correction.

5.1.6.1 Shaded Chopper

Task was about calculating the lighting using Phong lighting model. This was done with two shading models: the Gouraud (per-vertex, Illustration 54) and Phong (per-fragment, Illustration 53). Alternation between them was based on user input.

Students had to think about the vertex and fragment shaders, and see, how values are interpolated between them. Description of all the required shading and lighting models was also restated with a practical approach in the task description.

In order to see the specular highlight more clearly, the chopper's body was replaced with an approximation of a squashed sphere. There was also a question for the students, to explain, why there is no specular highlight in the back wall with Gouraud shading.

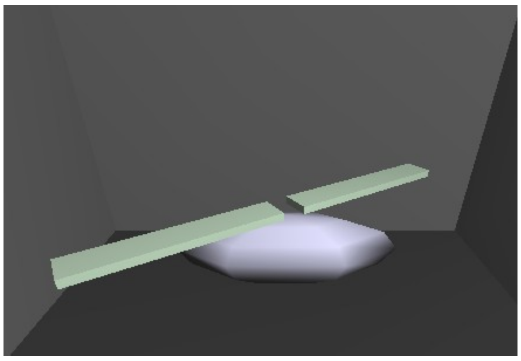


Illustration 54: Result of the Gouraud shading.

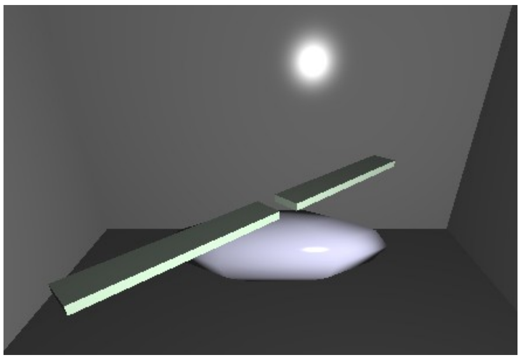


Illustration 53: Result of the Phong shading.

5.1.6.2 Blinn Chopper

The task described the Blinn's specular term and asked the students to change the result of the Phong shading from the previous solution to implement the Blinn lighting model.

5.1.6.3 Gamma Chopper

The task referenced the GPU Gems article on gamma correction [74] and asked the students to read that. After that, gamma correction (with $\gamma = 2.2$) was to be applied for the colors in the previous solution. Because the scene colors were configured for no gamma correction, then students were also first asked to do gamma decode on the inputs.

Results were described to have a more sharper edge between the directly illuminated and not illuminated areas (Illustration 55 and Illustration 56). The areas with no direct illumination were supposed to look lighter (because the ambient term would specify a higher value in sRGB after the gamma correction).

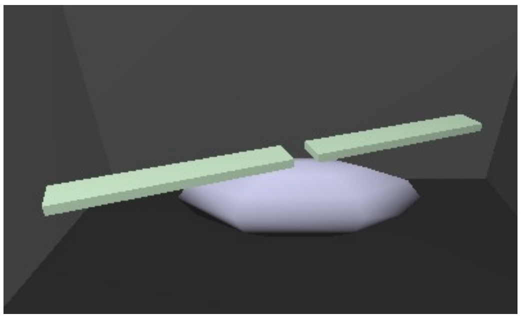


Illustration 55: Gouraud shading with gamma correction.

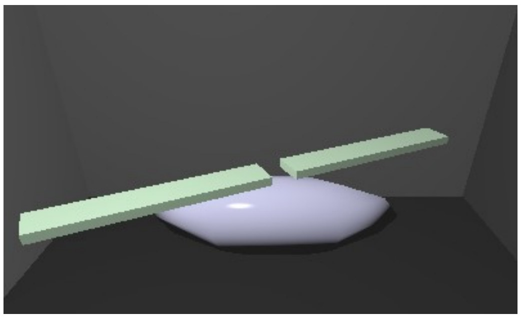


Illustration 56: Phong shading with gamma correction.

5.1.7. Textures and Sampling

The *Textures and Sampling* topic did not include the chopper scene in order to focus more on the described algorithms, and not the chopper.

5.1.7.1 UV Mapping

This task consisted of students specifying the UV mapping of a texture directly for the vertices of a plane and then using the interpolated UV coordinates to sample from a texture in the shader. The idea was to show that UV mapping an image to a quad is not complicated and give students the experience of using textures and sampling them in the shader. It was shown that we can interpolate different values between the shaders (not just *vec3*-s).

The result should have had the texture mapped to a quad such that the texture is repeated 2 times in both the horizontal and vertical directions. This caused the students to specify the repeat wrapping for the texture and think, how the UV coordinates need to be transformed to achieve the effect (Illustration 57).

TARTU ÜLIKOOL · UNIVERSITAS TARTUENSIS · 1632
Illustration 57: Image of the UT logo UV mapped to a quad in a way it repeats two times in both directions.

5.1.7.2 Bump Mapping

Here the technique of bump mapping was described together with finite difference schemes. Students had to pick one of the finite difference schemes to sample the bump texture in the shaders. The gradient vector, that was calculated based on a chosen scheme, needed to be applied to the surface normal and also used to sample the texture from a shifted position.

The idea was to have students see that not all textures are used to just specify the granular color of the surface. The resulting effect of this task was to have the surface changed lighting calculations and also present an effect that the texture is applied on top of the seemingly changed surface (ie deformed according to the deformations calculated from the bump map).

The task also included instructions to make the quad rotate around the y-axis. This would make it important to transform the gradient vector into the camera space, prior in applying it to the normal used in the lighting calculations. If the quad would have been still, then the x and y components of the gradient could have been made to match without the normal matrix.

Different textures and bump maps were provided in the task to allow students to experiment with simpler ones first (Illustration 58 and Illustration 59).

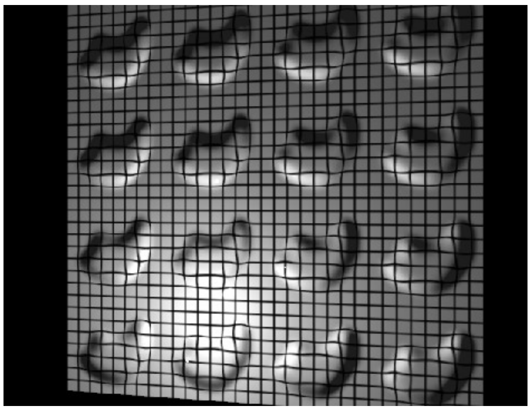


Illustration 58: Still image of the result with one of the test textures and test bump maps.

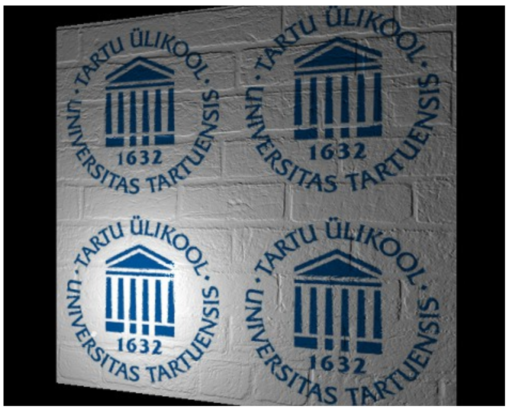


Illustration 59: Still image of the result with a more complex bump map and the UT logo texture.

5.1.8. Blending

The tasks in this topic were created by Jaanus Jaggo. The base code for the JavaScript was also created by him. The base code for C++ was created by me.

The idea of the tasks was to use the depth values of fragments to experiment with different blending in the context of a practical problem.

5.1.8.1 Soft Particle Chopper

Task consisted of the students adding smoke particles (transparent quads that had the smoke texture applied to them) to the chopper scene. The naive approach of this would have the texture clip be visible at the intersections of the quads and the scene objects (Illustration 60).

Instructions were given to have the transparency increase based on the nearness of the particle fragment's depth and the depth of the scene geometry at the same location. This would create soft particles that had no visible clipping with the scene geometry (Illustration 61).

The C++ code also introduced the need to sort the transparent objects in the scene based on their values in the z-axis of the camera, from far to near.

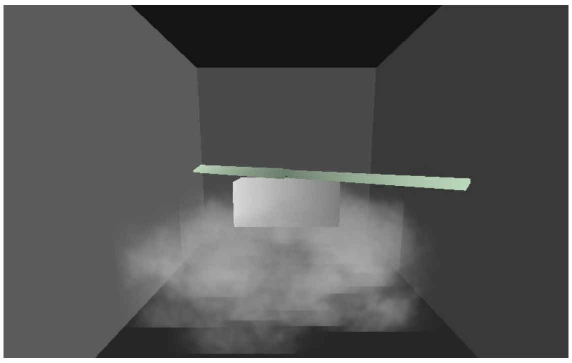


Illustration 60: Still image that shows the visible clipping of smoke particles with the scene geometry.

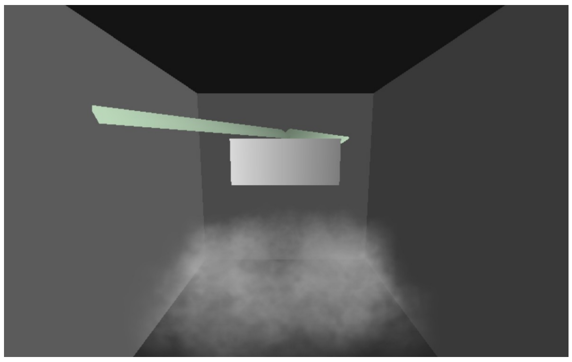


Illustration 61: Still image of soft particles that have their fragments' transparency increased based on the nearness to the scene geometry. No visible clipping.

*5.1.8.2 Custom B. Chopper**

This task asked the students to experiment with other blending modes to create a custom effect. The nature of the custom effect was left up to the students. An example was given that had the smoke texture applied to a quad in front of the camera. Blending was configured in a way, where the alpha value of the smoke texture indicated the opaqueness of the surface. The goal of this effect was described to the students to create a vignette or make it appear that the scene is looked through a dirty window (Illustration 62).

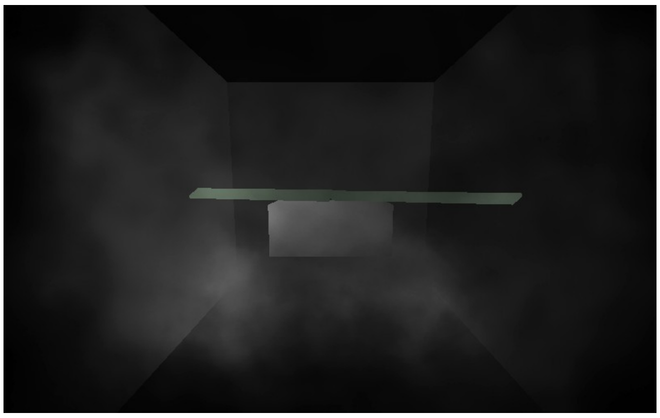


Illustration 62: Still of an example of the use of custom blending described to the students.

5.2. Basic II

Tasks in this module focus on a variety of computer graphics related topics. Those tasks assume that the material and tasks in Basic I were understood by students. For example, the Imported Chopper task assumes that a student understands the hierarchy in the object graph and knows, how it should have a logical structure that encapsulates the different parts of objects.

5.2.1. Modeling and File Formats

So far the students had created the objects in their scene using geometric primitives. This week's goal consisted of seeing and understanding the basic ideas behind different file formats that can be used to store 3D models.

5.2.1.1 *Imported Chopper*

Task requests the students to import a chopper model (modeled by Timo Kallaste) to the scene. The chopper comes in both the OBJ and Collada formats. The first one does not convey the hierarchy of the object, so students have to reconstruct that. The Collada format embeds the hierarchy data and students have to fetch the blades part of the chopper from that (in order to make it rotate).

There is also a model that includes animations. That model is taken from a Three.js example [75]. For the JavaScript task that model is in a JSON format native to Three.js. The model has animations for *idle*, *walk* and *run*. The task is to use the Bernstein polynomials to blend those three animations together, based on the movement speed of the character. This teaches students to handle animations in their application and also

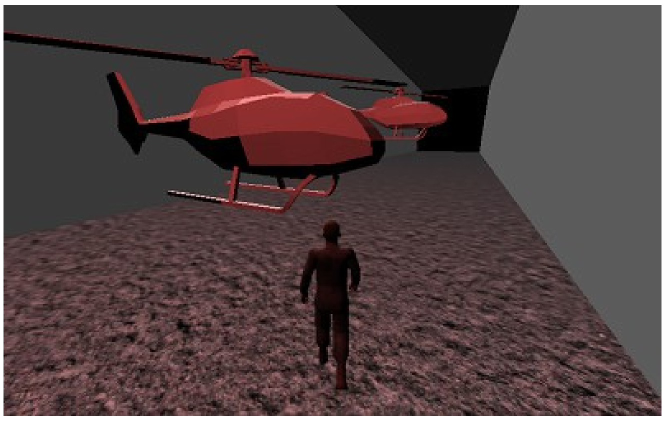


Illustration 63: Still image of the two imported choppers (OBJ and Collada formats) and an animated model of a marine.

introduces curves a bit.

5.2.2. Environment Mapping

Environment mapping teaches a way, how to take into account the surrounding environment, when considering the color of a surface. We focus on reflective surfaces here.

5.2.2.1 *Reflected Chopper*

This task asks the students to make the floor of the hangar reflective. The task consists of creating another camera and mirroring it from the floor. The scene rendered from that other camera should then be sampled to get the reflection values of the floor. This task is based on an idea described by Lauris Kaplisnki to create a reflective water surface [76].

Task should give the students a neat way to create a reflective surface. It recapitulates the usage of cameras (from the Projected Chopper task) and texture sampling (from the UV Mapping task).

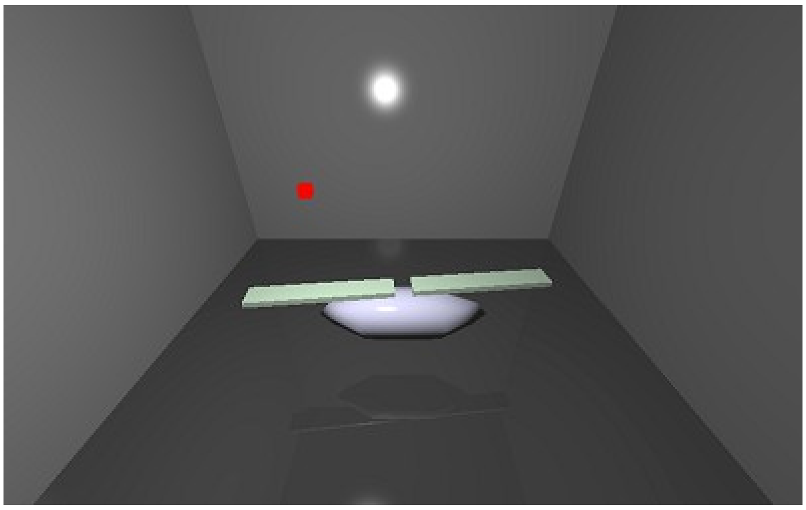


Illustration 64: Still image of the chopper in the hangar with a reflective floor. Red rectangle denotes the light source.

5.2.3. Curves

This week practices the construction of certain curves.

5.2.3.1 Bezier Spline

The task requires students to calculate correct control points for C^1- and C^2-smooth Bezier splines. Students are provided with a function (built in Three.js, added specifically for C++) that constructs a cubic Bezier curve, given 4 control points. The base code initially draws a C^0-smooth curve and students need to use the Stärk's construction ([77] and [78]) to create smoother splines.

Students should be able to relate to derivatives of a function, based on the instructions in the task description. This should show the importance of mathematically finding the derivatives of a function at certain points.

When the students have constructed the smooth Bezier splines, they are also asked to rotate the control points to illustrate the affine invariance of the Bezier curve.

Task's base code consists of two curves. One has randomly generated control points and the other is a test set of control points that specify the same Bezier spline as in [78].

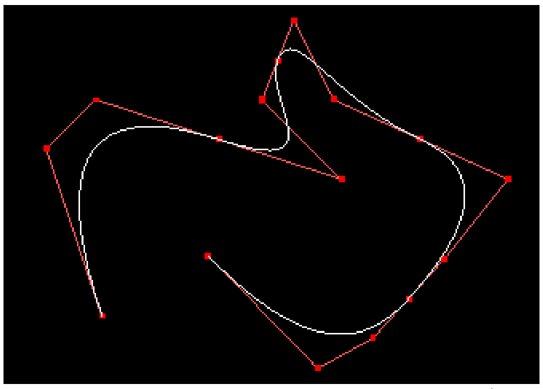


Illustration 65: Solution that has a C^1 -smooth Bezier spline with the test control points.

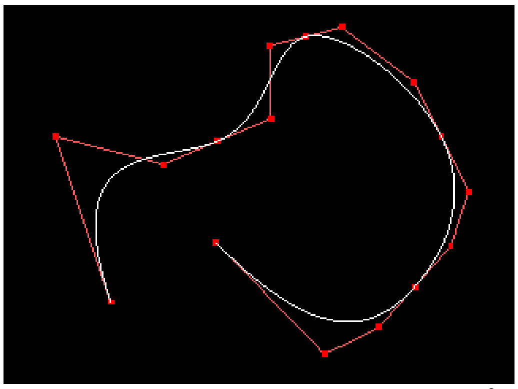


Illustration 66: Solution that has a C^2 -smooth Bezier spline with the test control points.

5.2.4. Procedural Generation

In this topic there are two tasks. The one on the use of Perlin noise was created by Jaanus Jaggo. There is another task on generating trees using Lindenmayer systems.

5.2.4.1 Perlin Planet

This task, created by Jaanus Jaggo, requires the students to generate a 3D Perlin noise on the GPU. This shows a bit different technique, than the one described in the corresponding material. The task is to color the surface of a sphere with 4 discrete colors, depending on the thresholded values from the noise (Illustration 67). The result also connects with the very first interactive example in the material, where there was a sphere, resembling the Earth, with Perlin noise used for the atmosphere. The technique here, using the Perlin noise as a height map, illustrates another application of it.

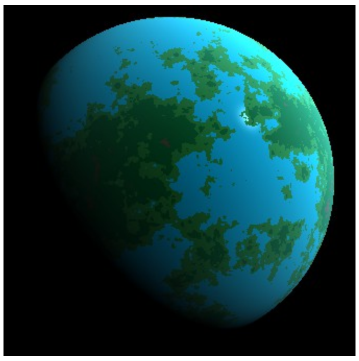


Illustration 67: Final result of the Perlin Planet task.

5.2.4.2 Lindenmayer Tree

Students are asked to implement a 0-context stochastic parametric Lindenmayer system. That system has only one stochastic rule:

$A \xrightarrow{0.(3)} F[+(a)FA][-(a)FA]A$

$A \xrightarrow{0.(3)} F[+(a)FA]A$

$A \xrightarrow{0.(3)} F[-(a)FA]A$

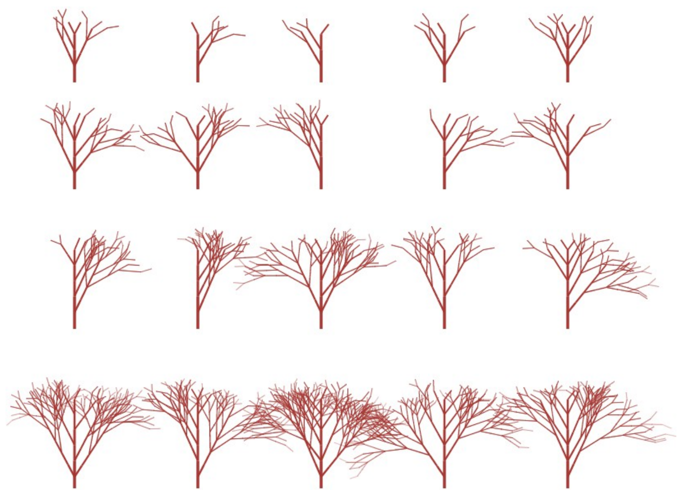


Illustration 68: Lindenmayer trees generated with the asked system. Different 5 results are shown for 4 varying number of iterations.

The task description explains the different symbols, their semantic meaning for the tree, and their turtle graphics interpretation. Students should generate words, based on those rules and starting from the axiom A, with a varying number of iterations (Illustration 68). Then the interpretation of the symbols by the turtle graphics approach is asked, in order to draw the trees. The result is a 2D tree, which means that the canvas drawing is used for JavaScript and the Allegro library for C++.

5.2.5. Ray Tracing, Space Partitioning, BVH

Although this topic has a title comprising of three subjects, there is only one task for ray tracing. Bounding volume hierarchy could be easily added to the ray tracing task. Space partitioning techniques would require of a lot more work.

5.2.5.1 Ray Chopper

Students are asked to finish a similar ray trace renderer that is shown under CGLearn's materials. The scene to be rendered is a similar chopper scene that was in the Cube Chopper task (meaning that the chopper consists of cuboids), although with actual light calculations (Illustration 69).

Students are asked to send a data texture to the shaders with the geometry. For each fragment, a ray is constructed and intersection with the geometry is tested via Möller-Trumbore ray-intersection testing algorithm [65], which the student should implement. For the closest intersection, the normal vectors also need to be fetched from the data texture. After this the light calculations are applied.

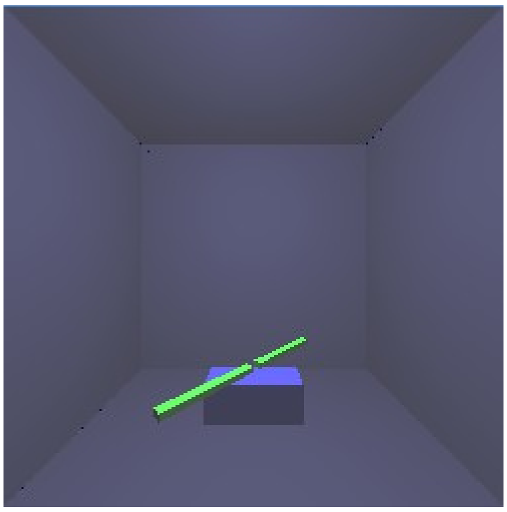


Illustration 69: The chopper scene rendered with a ray trace renderer.

5.2.6. Global Illumination

5.2.6.1 Path Trace

This task asks the students to build upon the code in the Ray Chopper task, in order to render a scene using path tracing with direct illumination, described in [67]. The task asks to implement a random reflect method that will create a random reflection vector given an incident vector and a surface normal. The technique of creating random vectors inside a cube and then picking those that also lie inside a sphere, is described and illustrated. If a random vector points away from the normal, then an opposite vector can be taken.

There is also a description, how to use two textures (one for rendering to, and another to sample from) in order to average the results between multiple rendering passes.

Students are asked to create a static scene instead of a moving chopper. That scene should illustrate indirect illumination. The task shows example illustrations that depict a scene with no bounces (only ray tracing and direct illumination, Illustration 70); the indirect illumination resulting from a bounce (Illustration 71); final result, where the indirect and direct illuminations are combined (Illustration 72).

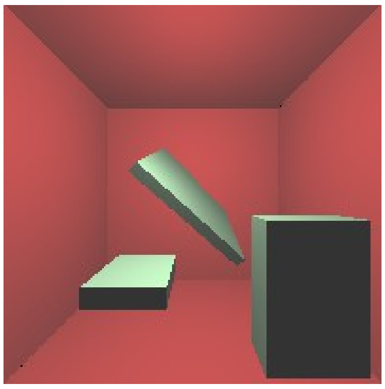


Illustration 70: Direct illumination only.

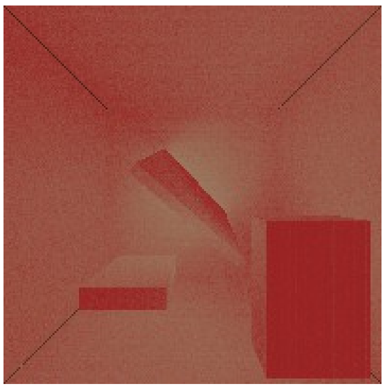


Illustration 71: Indirect illumination only.

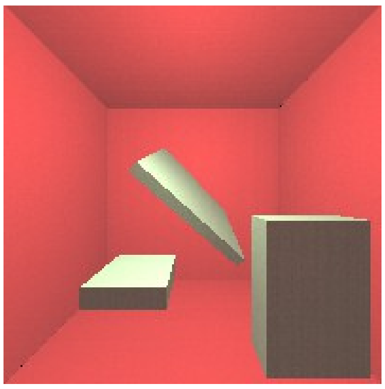


Illustration 72: Combined result.

5.2.7. Shadows

At the time of writing this thesis, the tasks in this final topic, were not yet completed.

6. The Flashcards

In order to allow students to test themselves on the covered material, a didactic tool of flashcards was implemented in CGLearn. Tool allows students to go through different terms and questions covered during the modules. Together with each question, there is also an answer on the other side of the flashcard. Students first see the question side (Illustration 98 in the Appendix) and think about the answer for the question. When ready, a student can turn over the flashcard and see the answer. After this, an estimate can be given specifying how well the student knew the answer (Illustration 99 in the Appendix).

The tool is based on the SuperMemo 2 algorithm [9] that takes into account the estimates given by individual students on individual flashcards. Based on the current and previous estimates by the student, a next display date for that flashcard is calculated.

Decks of flashcards are grouped under the Basic I and Basic II modules. The first deck includes 110 flashcards and the second 50 flashcards total. Such a grouping allows students to focus on the topics covered during one module, while still having a sufficiently large deck to provide a variation of topics.

6.1. Basic I

6.1.1. Computer Graphics

The flashcards created under this topic include a question about the areas of computer graphics use, and questions about different technologies: OpenGL, Allegro, WebGL, Three.js. There are also questions to name and describe different steps in the standard graphics pipeline.

6.1.2. Introduction to Geometry

Here there are questions about simple, non-simple, convex and concave polygons. Those questions depict a polygon (Illustration 73) and ask the student to categorize it. There is also a question to describe some geometric properties of a triangle.

Illustration 73: Polygon to categorize.

Next there are questions about different coordinates. Couple of questions denote a point in homogeneous coordinates and ask, what 3D point it represents.

There are also questions to find the Barycentric coordinates of some points on a triangle. Those depict a triangle in a Cartesian coordinate system and a point X on the triangle. Mostly the Barycentric coordinates can be derived just by their definition.

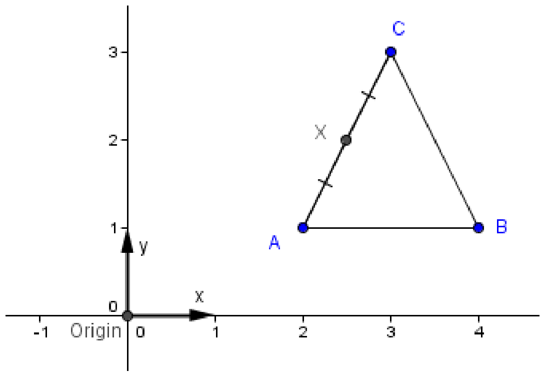


Illustration 74: The Barycentric coordinates of point X are asked.

Finally there are questions about simple vector operations like the dot product, cross product, vector-point addition and vector normalization.

6.1.3. Geometry and Transformations I

The questions here present a scale, rotation or shear matrices and ask the students to recognize them. Similar questions ask the students to construct a scale, rotation or a shear matrix.

There are also questions about matrix multiplication commutativity and associativity properties; and a question about the linearity of an transformation.

6.1.4. Geometry and Transformations II

Besides asking an analogous transformation questions about the translation transformation, the flashcards in this topic also asks to derive a matrix, given a depicted transformation. An illustration shows the standard basis and a transformed bases (Illustration 75). This question is about deriving the transformation matrix, using the basis vectors of the transformed standard basis as the columns, additionally adding the translation to the last column.

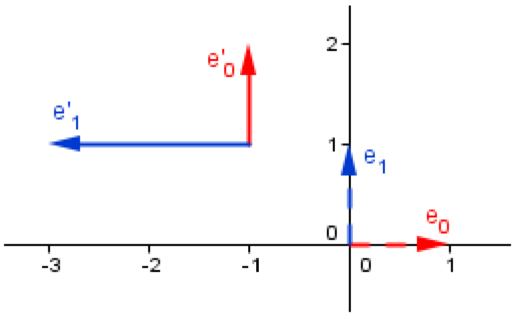


Illustration 75: Flashcard asks what matrix would do this transformation.

There are also two questions that depict an hierarchical object and ask about the transformations required for a specific node.

6.1.5. Frames of Reference and Projection

There are questions that ask an explanation for specific matrices: model, view, normal and projection. One question asks to name different frames of references that are used in graphics. Two questions ask, how to construct the camera's view matrix, given either the camera's model matrix or the *lookAt*, *up* vectors and the position.

Questions about projections ask about the different projections: orthographic, oblique and perspective. Couple of questions are about recognizing or constructing parts of the orthographic projection matrix.

6.1.6. Shading and Lighting

A number of questions ask the students to describe a color, which is given by its RGB values. The values of different channels are in different notations ($[0..1]$, $[0..255]$, hexadecimal) in different questions.

The notions of a color and ambient, diffuse, specular reflections are covered. The different reflections also have questions that ask, how to find them given a material, light source and a viewer. Specular reflection has two questions about it, one for Phong and other for Blinn-Phong lighting models. Three questions also ask about different shading models.

A couple of questions depict vectors originating from a point towards different objects. Those vectors are needed for light calculations and students are asked to name the direction of an indicated vector.

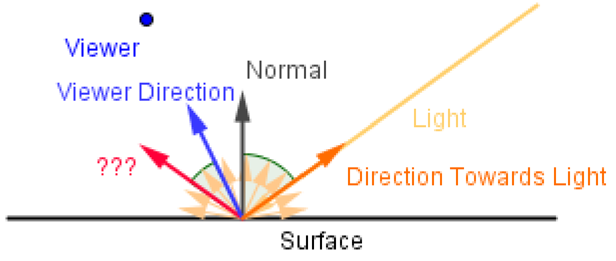


Illustration 76: Flashcard asks to name the indicated reflected light vector.

6.1.7. Textures and Sampling

This topic includes questions about different interpolation methods: nearest neighbour, linear, bilinear and trilinear.

There are also illustrations of certain texture sampling configurations that ask the students to guess them. Specifically students are asked to distinguish between no use of mipmapping that produces Moire aliasing (Illustration 77), and bilinear filtering with mipmapping that produces visible lines between mipmaps (Illustration 78).

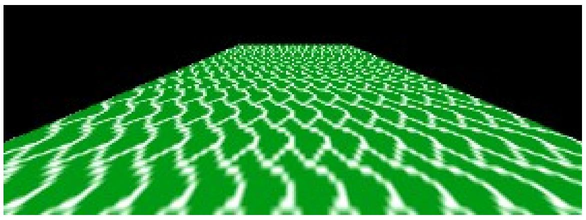


Illustration 77: No mipmapping is used, Moire aliasing is visible.

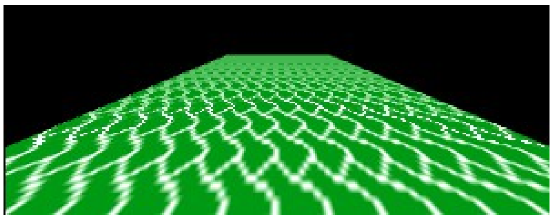


Illustration 78: Mipmapping is used, but lines between mipmaps are visible.

6.1.8. Blending

Questions here ask about different buffers: framebuffer, color buffer, depth buffer; and also about z-fighting. Two questions are about describing the conventional alpha blending and premultiplied alpha blending. Then the questions ask to construct the general blending function and different configurations of it.

6.2. Basic II

The flashcards in this module are about the general ideas described in the corresponding materials.

6.2.1. File Formats and Modeling

The questions are about different formats (OBJ, Collada, FBX) and their properties. Three questions ask directly about these formats. Two questions pose a problem of saving a described mesh, and ask about which format is preferable. One question is about general use of different formats.

6.2.2. Environment Mapping

Four questions here ask about the cube map, sphere map, sky box and sky dome. Two questions present images of a cube map and a sphere map (Illustration 79), and ask the student to recognize them. The images are based on the cube map by E. Persson that is also used in the interactive examples accompanying the Environment Mapping topic in the material.

Illustration 79: One flashcard asks to recognize this sphere map.

6.2.3. Curves

Questions about general terms include an approximating curve, interpolating curve, C^n and G^n smoothness, spline and blending functions. The answers to the approximating and interpolating curve have corresponding illustrations in them. Subsequent questions ask the student to recognize different curves and splines covered in the material. Those consist of recognizing an Hermite, Catmull-Rom (Illustration 80) and Bezier splines; B-Spline and NURBS curves.

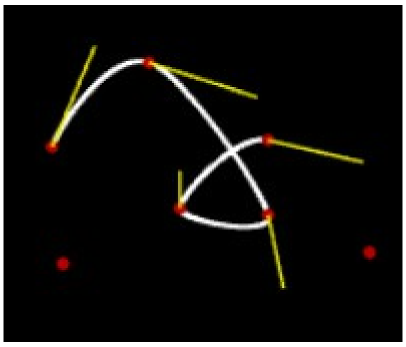


Illustration 80: One flashcard asks to recognize this Catmull-Rom spline.

6.2.4. Procedural Generation

Under this topic, there are creative questions, asking the student, what could be done with Perlin noise, Lindenmayer systems and particle systems. Answers include some general applications, but indicate that there are more uses. Next there are specific questions about procedural generation techniques. Those include asking about the iteration count in the Lindenmayer systems, the three main rules of the Boids algorithm and what does the Boids algorithm aim to emulate. The answer to the question about the rules of the Boids algorithm, shows an illustration of the rules (Illustration 81), together with individual descriptions.

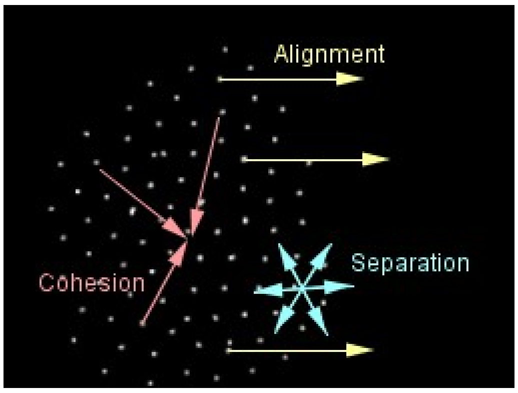


Illustration 81: Illustration on the answer side of the question about the rules of the Boids algorithm.

6.2.5. Ray Tracing, Space Partitioning, BVH

The first questions ask about the parametric representations of a ray and a triangle, and their use in the Möller-Trumbore ray-triangle intersection testing algorithm. Next, there is a creative question about the use of ray casting. Final two questions are about the performance of a ray trace based rendering and its optimizations.

6.2.6. Global Illumination

One question asks the students to recognize the Cornell Box scene from an image (Illustration 82). Next, a description of the path tracing algorithm is given, and students are asked for the name of the technique. Similarly one question asks to name two global illumination algorithms. The answer to that question, provides the names of three techniques. Two questions include the terms local illumination and global illuminations.

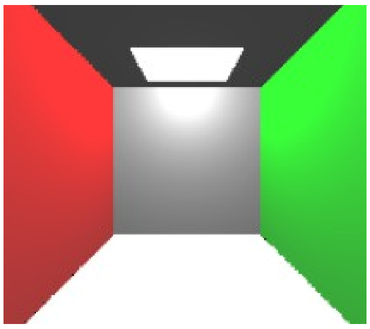


Illustration 82: One question asks to recognize this Cornell Box scene.

6.2.7. Shadows

Three questions ask the students to describe the umbra, penumbra and antumbra parts of a shadow. There are two questions about the umbra and penumbra that ask to recognize them from an image (Illustration 83). One question asks, which of the three given light sources (directional, point, area) produce a penumbra.

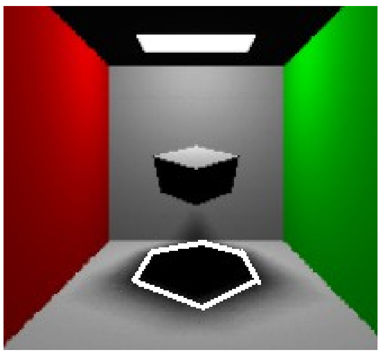


Illustration 83: Students are asked to name the indicated part of the shadow.

Next, are the questions about different techniques. These include a question to name two common shadow rendering techniques. Answer to this question provides three (projection shadows, shadow mapping, shadow volume). Two questions ask about the main problems with shadow mapping and shadow volume algorithms. One question asks about shadow rays, namely how to detect a shadow in a ray trace based rendering.

This concludes the overview of the flashcards created for the students. Overall goal was to reinforce the ideas from the material and lectures. Thesis proceeds with the results of using the material described so far and a discussion of those results.

7. Results and Discussion

The usefulness of this material and the CGLearn learning environment was assessed via a feedback questionnaire. At the time of writing this thesis, the Computer Graphics course was still in progress. This means that the feedback was given by the students prior to completing the course and thus may reflect assessments of partial material. Secondary feedback is planned to be asked during the exam, at which point the students should have a complete experience of the material and CGLearn.

The lectures were additionally assessed, in the form of exit cards described by Karm in [13]. The first question "What did you learn today?" aims to ask the students to describe, how well did they understood the material presented in the lecture. Answers to this question from the students, describe the quality and clarity of the lecture.

Students registered to the course (25 in total) were from different curricula (Table 2). The attendance of lectures and responses to the questionnaire also varied. Initially there were 50 students registered, but by the time of writing this thesis, 25 had unregistered.

Table 2: Overview of the curricula of students registered to the Computer Graphics course in spring 2015. Ordered by the number of students per curriculum, then by the levels of study and year.

Curriculum	Level of Study	Year	Count
Computer Science	Bachelor's	1	1
		2	5
		3	4
	Master's	1	6
Software Engineering	Master's	1	1
		2	1
Physics	Master's	1	1
	Doctorate	2	1
Mathematics	Bachelor's	1	1
Information Technology	Bachelor's	3	1
Teacher of Mathematics and Informatics	Master's	1	1
Engineering and Technology	Doctorate	1	1
Zoology and Hydrobiology	Doctorate	3	1

7.1. Questionnaire

The questionnaire, comprising of 28 questions, was sent to the participants of the course on 19.04. Out of the 28 questions, 19 were quantitative and 9 textual. The questionnaire was structured into 3 parts that asked about lectures, practice sessions and the CGLearn environment. Participants had until 09.05 to fill in their answers. Both an online form and a printed out questionnaire were used to gather results.

In the time period given to answer the questionnaire, 9 students filled in the online form, and 1 preferred the printed out version. Generally the results indicated that the course and its materials were really good. Some even mentioned that it is one of the best courses they had taken in the University of Tartu. There were also comments about quite different aspects that individual students found problematic.

Results and the questionnaire are in the corresponding folder accompanying this thesis. The box plots of the quantitative question answers are shown in Illustration 100, Illustration 101, and Illustration 102 in the Appendix. Box plots in those illustrations show the extreme values at the end of the whiskers and the first and third quartiles are the start and end of the box. This means that the estimations, which received at least 50% of the answers, are covered by the box in the box plot.

7.1.1. Lectures

The attendance of lectures among the participants of the questionnaire was high: 8 students attended 75%-100% of the lectures; 2 students marked the attendance 25% and 0%. Those students, who attended the lectures, also found the traditional lectures very useful. The Recapitulation and Conclusion lectures were not found that useful, with the average usefulness estimated overall around 55%. The fact that some lectures were conducted by different people was considered to be also on average 55% useful.

The textual answers explained the problems regarding the low participation of the students, who answered the questionnaire. One student expressed that the lecture is not a suitable format for learning, he or she learns more via individual practice rather than discussion.

Lectures were praised for the inclusion of a lot of examples (including the interactive examples from CGLearn); good use of the blackboard; openness to questions and discussion; involvement of students via questions about the material; use of the exit cards.

Critique was about the fact that some people do not like being asked questions about the material, tempo of the explanations was a bit too fast, blackboard illustrations were sometimes too unclear, few topics seemed to not fit into the course or were not accompanied by a practical task, slides were considered to have too few pictures. One student did not understand the usefulness of the second question (What more would you like to know?) on the exit card. One of the guest lectures was assessed to have not been very good.

Some of the critique is objective and should be taken into account for further conductions of the course. For example, the use of the blackboard can certainly be improved to include more clear illustrations. Tempo of the explanations can also be a bit corrected, but often the lectures finished exactly on time, or even a couple of minutes later. This means that if the lecture were conducted in a slower pace, then some of the material would need to be left out of the lecture. Because some students found the lecture to be more understandable, if they had previously read the material in CGLearn, then this form of study should perhaps be more advertised, so that students can better follow the lectures.

Other critique seems to be not that objective. For example, when talking to some of the students, none of them found that the slides had too few pictures. Rather the opinion was contrary, there were always illustrations and pictorial examples following most of the concepts.

The problem that some students disliked the questions and discussions in the lecture is a challenging one. The initial approach was that if a student did not know an answer, then a discussion is required in order to understand, where the student is lacking the understanding that keeps him or her answering. This discussion would find out the current level of the student, and help to build on top of that to reach the level required for the question. This is contrary to other lectures, where often students are required to study on their own if they are not on the same level as the lecture. Because lectures provide a possibility for a two-way communication between the student and the educator, then the first approach would initially seem to be preferable to both. Although in the light of the feedback, this might merit reconsideration.

7.1.2. Practice Sessions and Homework Tasks

7.1.2.1 Practice Sessions

The attendance of practice sessions was high among the students, who participated in giving the feedback: 9 of the students participated from 75% to 100% of the sessions; only one student estimated the participation at 25%. The average usefulness of the practice sessions was estimated at 75%. Although, the average is higher then the average usefulness of the lectures, more students chose 75% and 50% estimations. The fact that some of the practice sessions were conducted by different people, was estimated to have 70% usefulness and the quartiles were roughly the same.

The textual feedback reflected some contradicting opinions. The practice sessions were praised for the fact that we started doing the homework tasks in the classroom. This meant that students had to spend less time getting to know the task and the base code. Students considered the explanations given by the instructors to be very important. Sessions ran smoothly and were engaging for the students. The base code was also praised for its existence and comments in it. Some students found that the tasks were relatively independent, which meant that they did not have to have the previous tasks done in order to continue.

The critique from some students was about the dependencies of the tasks. Some students found that the tasks should not be that independent and there should be a couple of bigger tasks that get continued in each of the sessions. Other students felt that the tasks were too dependent, and they found it hard to complete new tasks without doing the previous ones. One student felt that the explanations given in the practice session restricted his or her creativity and that it would be better to first have the students to try to solve the task themselves. A couple of students found the explanations hard to follow. One reason for that was the late time (16:00 – 18:00) of one of the practice sessions and the other was that it is easy to sidetrack into debugging your own code, thus missing the subsequent explanations. One of the first practice sessions felt rushed for one of the students, and there was also critique for one of the instructors, who did not answer to student's e-mails and took a lot of time to grade the solutions.

The practice session start time could certainly be improved (moved to an earlier time).

Although, there was another practice session (for the Basic I and Game Engines modules) that did start at 12:00. For some reason that earlier time found a really low participation rate. This might indicate that an early time might not be suitable for other students.

Some of the tasks did have a dependency with previous tasks, but those were often few in number. Although some of the tasks were with a similar layout, there was no direct dependency between them, and other students even liked the similarities and the fact that some of the tasks gave a chance to recapitulate ideas from before.

Almost all of the tasks were available at least 2 weeks before the corresponding practice sessions (ideally they would have been available at the start of the course). This should have given the students enough time to try to solve the tasks themselves, and ask help in the practice sessions, if needed. It is hard to imagine, why the students did not exercise the possibility to try and solve the tasks beforehand (if they so pleased).

7.1.2.2 Homework Tasks

The percentage of solved homework tasks among the participants of the feedback was also high: 8 students had solved from 75% to 100% of the tasks; 2 had done 25%, among those 1 student commented that he or she is still doing the tasks. The modular layout of the course was considered to be quite helpful, with the average estimation of 77.5%. The base code of the tasks received an average of 95% helpfulness estimation. The task deadlines (one in the middle of the semester, and one in the end) for all the tasks, received mixed feelings from the students, and got an estimated 77.5% average helpfulness. The possibility to resubmit the solutions prior to the deadline was considered very helpful, and received 95% average estimation.

In the textual answers, many students praised the two deadline approach, and mentioned that they had some busier weeks, where they would have otherwise not been able to submit solutions. The base code and the comments (that specified step-by-step things needed to be done) in it were also praised. The ability to resubmit and the feedback given to the different solutions was considered to be really good. Students mentioned that this encouraged them to fix the problems in their solutions and better understand the problem.

One student found that the descriptions of the tasks did not state the task requirements clearly enough. A couple of students found that the connection between the tasks and the corresponding lecture might have been too weak. Another student found that the tasks covered a variety different topics, and thus required him or her to understand too many new ideas.

One student found that he or she still does not feel competent in JavaScript and Three.js, and thus finds it hard to understand the logic behind the code. Another student considered that maybe the possibility to solve the tasks in different languages is too confusing for students.

There was critique about the two deadline system from one student. He or she felt that the lack of time management skills made him or her not being able to solve the tasks for the deadline.

The feedback for the task description's clarity and base code's correctness is certainly something to take into account. On the other hand, lot of the textual feedback seems to be contradicting and indicates that some of the students misunderstand their possibilities. It was often mentioned by the instructors that ideally the tasks should be solved in the corresponding week. Doing so, also opened up the possibility to submit a fixed solution, thus rewarding the ones who did submit the solutions in sensible time. Time management is one of the skills that each student should possess. It is hard to say, why some students start solving the tasks proportionally too late, thus also overestimating their skills. While weekly deadlines would certainly alleviate that problem, they would, in fact, manage only the symptoms and not help students to develop correct time and skill estimation for themselves. The CGLearn environment even provides average time spent on each task, so that students can better estimate, how much time they would need to solve it.

7.1.3. CGLearn Environment

The usefulness of the functionality in the CGLearn environment was very highly rated (Table 3).

Flashcards and statistic charts got the lowest estimations. Regarding the flashcards, one student commented that he or she has not used them yet. Because this questionnaire was sent to the students before the end of the course, this result might not reflect the true usefulness of that functionality. Although students are told to go through the flashcards every couple of weeks, some of them will do it only on the weeks before the exam.

Table 3: Average estimates of the usefulness of CGLearn's functionality.

Functionality	Average
Written material	80%
Interactive examples	85%
Flashcards	70%
Task tree	97.5%
Task descriptions	97.5%
Task feedback	97.5%
Statistic charts	70%
Results table	85%

The usefulness of the statistic charts is expected. Because of the different personalities of the students, some might be more competitive or analyzing than others. For those, who need a general overview of the task difficulties, time estimates and the scores received, the statistic charts are more useful.

In the textual feedback, couple of students praised the environment, and stated that it is one of the best environments, they have used for learning. The average time spent in the environment per week was about 2 hours. This is excluding one student, who indicated that he or she spends more then 8 hours per week in CGLearn.

7.2. Exit Cards

The last 5 – 10 minutes at the end of most of the lectures, were dedicated to students answering two questions about the covered topic:

- What did you learn today?
- What more would you like to know?

One purpose of those questions, was to give feedback for the lecture's quality. Answering the first question, shows how much of the new material students managed to understand. If the student did not mention a topic covered in the lecture, then it might have been:

- already known,
- left unclear,
- hard to formulate.

The number of cards that included a certain topic can be regarded as a lower bound on the number of students who learned it. Not mentioning a topic does not necessarily indicate not learning it.

The second question gave a possibility to either ask additional questions about the covered material, or propose new ideas, what to cover. All questions and misunderstandings were addressed in the mailing list during the week following the lecture.

Next, the thesis provides an overview of the collected exit cards and the analysis of the first question. In the time of writing this thesis, not all of the lectures have been conducted.

7.2.1. Introduction to Computer Graphics

In the end of the first lecture, 33 exit cards were written. The Table 4 lists the common concepts that students mentioned in the answers to the first question.

Table 4: Overview of the common answers in the exit cards for the lecture Introduction to Computer Graphics.

Term, idea, concept	**Count**
Course organization	11
Coordinate system handedness	4
Polygons (convex and concave)	9
Triangle usefulness, vertex order, faces	10
Standard graphics pipeline	8

Students also mentioned that some of the terminology was new to them. Especially English terminology and asked, if the Estonian translations could be added to the slides. This also reflected from other answers, where students were unable to write *polygon*, and called it a *thingy* or *fancy word*.

A couple of students noted that they did not learn anything new and indicated that most of the material they were well aware of. One student described that it would have been better for him to understand the material, if it would have started from vertices and then gradually moved to edges, faces and polygons.

Many students liked that the course layout and organizational aspects were described in detail.

7.2.2. Introduction to Geometry

The second lecture produced 18 exit cards.

Table 5: Overview of the common answers in the exit cards for the lecture Introduction to Geometry.

Term, idea, concept	Count
Points and vectors	10
Convex combination	6
Barycentric coordinates	6
Vector operations (dot, cross, box product)	4
Vector normalization	2

Some of the responses mentioned that most of it was recapitulation of the algebra and geometry, they had learned in other courses before. Few indicated that it was good that the algebra and geometry was described in the computer graphics context.

Two of students learned about vector normalization, and did not write anything else. This may indicate a lack in the previous knowledge in algebra and geometry.

7.2.3. Geometry and Transformations

The third lecture had 13 exit cards.

Table 6: Overview of the common answers in the exit cards for the lecture Geometry and Transformations.

Term, idea, concept	Count
Transformations	8
Homogeneous coordinates	2
Matrix stack	1

A large number of the exit cards mentioned that they learned about transformations, although it varied, what exactly about the transformations was new. Several cards indicated that they learned to distinguish between linear and affine transformations.

7.2.4. Frames of Reference and Projection

This lecture had 8 exit cards. The number of students participating in the lectures, had decreased a lot.

Table 7: Overview of the common answers in the exit cards for the lecture Frames of Reference and Projection.

Term, idea, concept	Count
Frames of reference (different spaces)	6
Projections (orthographic, perspective)	5

Most of the cards mentioned different spaces and different projections. A couple of cards mentioned that the derivation of different transformation and projection matrices was learned, but needed some time and experience to really understand.

7.2.5. Shading and Lighting

Fifth lecture produced 10 exit cards.

Table 8: Overview of the common answers in the exit cards for the lecture Shading and Lighting.

Term, idea, concept	Count
Shading models (flat, Gouraud, Phong)	4
Lighting models (ambient, diffuse, Phong)	6
Color spaces (sRGB)	5

Almost all the cards indicated that students learned something new about the shading and lighting models. In a couple of cases, it was not exactly worded, what did they learn. One card consisted only of a question about projects, and did not answer the questions about the material. This indicates that often students do have questions and will write them down, given a chance, but are reluctant to ask them in person.

7.2.6. Textures and Sampling

This lecture had 9 exit cards.

Table 9: Overview of the common answers in the exit cards for the lecture Textures and Sampling.

Term, idea, concept	**Count**
Textures (incl mapping)	3
Texture sampling / filtering	5
Mipmapping	3

Two of the cards stated that the students now are aware what some of the graphics options in computer games actually do. Number of cards mentioned texture scaling, it is a bit unclear, if this means that the student learned about up-scaling, down-scaling, interpolation techniques and mipmapping, or just some of them.

7.2.7. Blending

The Blending lecture had 8 exit cards.

Table 10: Overview of the common answers in the exit cards for the lecture Blending.

Term, idea, concept	**Count**
Depth buffer	4
Conventional and premultiplied alpha	2
Blending modes	4

Couple of the cards stated that they had been using the covered techniques for a while, but did not know how they actually work.

7.2.8. Curves

There were 7 exit cards in the Curves lecture.

Table 11: Overview of the common answers in the exit cards for the lecture Curves.

Term, idea, concept	**Count**
Curves	2
Splines	2

Some of the cards only stated that they did not know much about the curves before, and learned lots of new formulas. Although, these cards did not mention what exactly those new formulas were and how did the students understand them. In contrast, another card mentioned that most of the mathematics was just a review for him or her. One card mentioned that the explanation in the lecture was complicated, and the student did not understand how to use the blending functions to draw the curve. On another card it was written that the student now understood where the Bernstein polynomials in the Imported Chopper task came from.

One of the cards asked about clothoid splines. That question was answered in the mailing list afterwards.

7.2.9. Procedural Generation

The Procedural Generation lecture produced 5 exit cards.

Table 12: Overview of the common answers in the exit cards for the lecture Procedural Generation.

Term, idea, concept	**Count**
Noise	3
Perlin noise	2
Lindenmayer system	1

Some of the cards just mentioned noise, others specified that they learned about Perlin noise in particular. Some of the cards mentioned that they were reminded about the concepts learned in the Automata, Languages and Compilers (MTAT.05.085) course. Only one of the cards mentioned Lindenmayer systems for generating trees. One student wrote that it is hard to understand the questions that another student asked in the lecture.

7.2.10. Ray Tracing, Space Partitioning, BVH.

The Ray Tracing lecture had 5 exit cards.

Table 13: Overview of the common answers in the exit cards for the lecture Ray Tracing, Space Partitioning, BVH.

Term, idea, concept	Count
Rays	4
Ray-triangle intersection	1
Space partitioning	2

One card specified that the Möller-Trumbore ray-triangle intersection algorithm was a bit complicated for the student. That student asked about that after the practice session, where the algorithm was explained again, and the student then understood it. Two of the cards were very general, and told that some parts of the lecture were not understood (without specifying the exact topics that were left unclear).

7.2.11. Global Illumination

In the Global Illumination lecture there were 4 exit cards.

Table 14: Overview of the common answers in the exit cards for the lecture Global Illumination.

Term, idea, concept	Count
Ray trace based global illumination	2
The Rendering Equation	2

One card specified that the student learned about the reflection of light. Another mentioned just generally having learned about global illumination techniques and mentions that it would be good to see those algorithms in practice. There was a question about global illumination techniques in real-time games. One of the practice sessions in the Game Engines module will answer that question.

This concludes the overview of exit cards received in the time of writing. Overall this technique benefits both the students and educators. It helps students to systematically formulate new concepts, provides feedback for the educator and serves as a communication channel between them.

8. Summary

This thesis described the work done for an elective, 6 credits Computer Graphics (MTAT.03.015) course aimed at the Master and Doctoral students in the Computer Science curriculum. The thesis started with the description of the mentioned course and explained the proposed modular layout for it. Next it was described, how the course was conducted using a custom learning environment CGLearn, which is the main result of this thesis.

In the second chapter, titled CGLearn Learning Environment, the requirements for the environment were written. Those requirements were compared against the functionality of the previously existing and used solutions (Courses page, Moodle, Udutu). It was concluded that none of the current solutions would fill the requirements with an acceptable amount of effort. Thus the realization of a custom environment was accepted. The thesis described the technical choices and implementation details of that custom environment. These were described for both the back-end and front-end. For the former, the controllers and services in the *Student* and *Teacher* modules were described. For the front-end, the thesis first described the different JavaScript modules that were implemented and used. After this, there were descriptions of the more complicated interactive examples inside the material. The last part of the CGLearn chapter described the actual functionality available for both the student and teacher users of the system. This included navigating and reading the material, submitting homework task solutions, grading them, seeing the overall score achieved.

The Materials chapter talked about the written material and the interactive examples used in the course. The thesis focused on the Basic I and Basic II modules (together they included 15 different topics), because the majority of the material in those were done as the work of this thesis. The main goals of the material and the purposes that the interactive examples served were described.

The Tasks chapter proceeded in describing the homework tasks in the Basic I and Basic II modules. There were 22 tasks created in the scope of this thesis. In total there were 26 tasks (at the time of writing this thesis), 4 of them were created by other authors. Thesis described the goals of the tasks, and the skills students should develop by solving the tasks.

The Flashcards chapter described the flashcards in CGLearn. There were 110 for the Basic I module and 50 for Basic II. Flashcards for each of the topics were explained.

The seventh chapter, titled Results, first described the students, who participated in the course. The results of the material and environment were assessed via a questionnaire. Those results, as given by 10 of the participating students, were described in that chapter, together with the analysis of the textual answers. An overview of the lectures was given via the results of the exit cards. Exit cards were given to the students in the end of each lecture, and they asked, what the students had learned and what else would interest them. Results of mostly the first question are described in this thesis.

Reflecting on the work done, I conclude that the designed and implemented custom learning environment CGLearn was quite useful for the students of the Computer Graphics course. This is also conveyed in the results of the feedback questionnaire. From an educators perspective, the creation of the environment together with the material and interactive examples was sufficiently easy. The estimated time spent on that work by me fits well into the requirements of a Master's thesis.

Future development of the environment is planned, and is currently on hold, because the current course is still running and students are using the environment. In order to avoid unexpected system failures that might hinder the current users, the subsequent development will start after the course has finished. Further development includes a preparation of the system for a next Computer Graphics course; creation of publicly available pages with some parts of the material; inclusion of another server for file (primarily the students' submissions) storage; fixes of known deficiencies.

First, thanks to all the students, who took part in this experimental course. Thanks to other educators: Margus Luik, Jaanus Jaggo, Ats Kurvet, Timo Kallaste and Benson Muite; who helped me conduct the course, *e.g.* gave some of the practice sessions and lectures, previewed the lecture slides, proposed suggestions and help. Thanks to Indrek Kõnnussaar and his company (Blue Lynx OÜ) server for hosting the CGLearn environment and providing timely support always. Of course I need to thank my supervisor Konstantin Tretyakov, who took the time to explain a number of questions I had about computer graphics concepts and for conducting the Computer Graphics course last time. Thanks also to Ilya Kuzovkin – the other conductor of the previous course – for inspiration. I thank my other supervisor Anne Villems for giving me feedback and recommendations on how to structure this thesis. Lastly, I thank the University of Tartu and the Institute of Computer Science for letting me use this material and CGLearn in a real course.

9. References

[1] Stanford University, CS 148: Introduction to Computer Graphics and Imaging (Fall 2014), http://web.stanford.edu/class/cs148/ (14. 05. 2015).

[2] Department of Computer Science, Columbia University, Vision & Graphics, http://www.cs.columbia.edu/education/ms/visionAndGraphics (14. 05. 2015).

[3] Institute of Computer Science, University of Tartu, Computer Graphics (MTAT.03.015), https://www.is.ut.ee/rwservlet?oa_aine_info.rdf+1006870+HTML+92522999797432876337+text/html (14. 05. 2015).

[4] Avi C. Naiman, Interactive Teaching Modules for Computer Graphics, ACM SIGGRAPH, 1996.

[5] R. Klein, F. Hanisch, W. Straßer, Web-Based Teaching of Computer Graphics: Conceptsand Realization of an Interactive Online Course, ACM SIGGRAPH, 1998.

[6] J. Tomanová, M Cápay, E-learning Support for Computer Graphics Teaching and Testing, TELE-INFO, 2010.

[7] TÜ haridustehnoloogiakeskus, Tartu Ülikooli Moodle´i õpikeskkond, https://moodle.ut.ee/ (14. 05. 2015).

[8] Institute of Computer Science, University of Tartu, Arvutigraafika / Computer Graphics, 2013 Fall - Courses page, https://courses.cs.ut.ee/2015/cg/spring (14. 05. 2015).

[9] P. A. Wozniak, SuperMemo 2: Algorithm, http://www.supermemo.com/english/ol/sm2.htm (14. 05. 2015).

[10] E. Angel, S. Cunningham, P. Shirley, K. Sung, Teaching Computer Graphics withoutRaster-Level Algorithms, ACM SIGCSE Bulletin, 2006.

[11] Kelvin Sung, Peter Shirley, A Top-Down Approach to Teaching Introductory Computer Graphics, ACM SIGGRAPH, Educators Program, 2003.

[12] Institute of Computer Science, University of Tartu, Arvutigraafika / Computer Graphics, 2014 Springl, https://courses.cs.ut.ee/2015/cg/spring/ (14. 05. 2015).

[13] Mari Karm, Õppemeetodid kõrgkoolis, 2013, Sihtasutus Archimedes.

[14] D. R. Krathwohl, A revision of Bloom's taxonomy: An overview, Theory Into Practice, vol 41, no 4, 2002.

[15] Naps et al, Exploring the Role of Visualization and Engagement in Computer Science Education, ACM SIGCSE Bulletin, 2003.

[16] Google, Chrome Browser, https://www.google.com/chrome (14. 05. 2015).

[17] The PHP Group, Alternative PHP Cache , http://php.net/manual/en/book.apc.php (14. 05. 2015).

[18] J. Burke, K. Westin, K. Harsh, M. Medeiros, RequireJS Optimizer, http://requirejs.org/docs/optimization.html (14. 05. 2015).

[19] P. R. Michaud., PmWiki, http://www.pmwiki.org/ (14. 05. 2015).

[20] J. Vajakas, A. Lissitsin, M. Johanson, S. Laur, D. Unruh, P. Laud, MathWiki, http://mathwiki.cs.ut.ee/ (14. 05. 2015).

[21] Udutu, Udutu, http://www.udutu.com/ (14. 05. 2015).

[22] Udutu, FAQs, http://www.cedarlearning.com/resources_faqs.html (14. 05. 2015).

[23] T. Otwell, Laravel Framework, http://laravel.com/ (14. 05. 2015).

[24] Doctrine Team, Doctrine Project, http://www.doctrine-project.org/ (14. 05. 2015).

[25] Internet2 Consortium, Shibboleth, https://shibboleth.net/ (14. 05. 2015).

[26] J. Burke, K. Westin, K. Harsh, M. Medeiros, RequireJS, http://requirejs.org/ (14. 05. 2015).

[27] The jQuery Foundation, jQuery, https://jquery.com/ (14. 05. 2015).

[28] M. Otto, J. Thornton et al, Bootstrap, http://getbootstrap.com/ (14. 05. 2015).

[29] R. Cabello et al, Three.js, http://threejs.org/ (14. 05. 2015).

[30] D. Cervone, V. Sorge, C. Perfect, P. Krautzberger, MathJax, https://www.mathjax.org/ (14. 05. 2015).

[31] Highsoft AS, Highcharts, http://www.highcharts.com/ (14. 05. 2015).

[32] J. Skarnelis, FancyBox, http://fancybox.net/ (14. 05. 2015).

[33] F. Heinze, Laravel vs. Zend Framework 2 comparison | vsChart.com, http://vschart.com/compare/laravel/vs/zend-framework (14. 05. 2015).

[34] Creolab et al, Laravel Modules, https://github.com/creolab/laravel-modules (14. 05. 2015).

[35] M. van Wijngaarden et al, Doctrine 2 for Laravel, https://github.com/mitchellvanw/laravel-doctrine (14. 05. 2015).

[36] Linode LLC, SSD Cloud Hosting - Linode, https://www.linode.com/ (14. 05. 2015).

[37] Canonical Ltd., Ubuntu, http://www.ubuntu.com/ (14. 05. 2015).

[38] The Apache Software Foundation, The Apache HTTP Server Project, http://httpd.apache.org/ (14. 05. 2015).

[39] Blue Lynx OÜ, Codelight, http://codelight.eu/ (14. 05. 2015).

[40] University of Tartu, Study Information System, http://ois.ut.ee (14. 05. 2015).

[41] M. Bouroumeau-Fuseau et al, PHP Debug Bar, https://github.com/maximebf/php-debugbar (14. 05. 2015).

[42] Oracle Corporation, MySQL Workbench, Oracle Corporation (14. 05. 2015).

[43] Atlantic18 et al, Doctrine2 Behavioral Extensions, https://github.com/Atlantic18/DoctrineExtensions (14. 05. 2015).

[44] Doctrine Team, Inheritance Mapping: Class Table Inheritance, http://doctrine-orm.readthedocs.org/en/latest/reference/inheritance-mapping.html#class-table-inheritance (14. 05. 2015).

[45] B. Nesbitt et al, Carbon, https://github.com/briannesbitt/Carbon (14. 05. 2015).

[46] S. Petre et al, Slider for Bootstrap, http://www.eyecon.ro/bootstrap-slider/ (14. 05. 2015).

[47] StackOverflow, Mathjax equations displaying 3 times (Jekyll), http://stackoverflow.com/questions/28726152/mathjax-equations-displaying-3-times-jekyll (14. 05. 2015).

[48] S. Smith, Modular HTML components with RequireJS, http://simonsmith.io/modular-html-components-with-requirejs/ (14. 05. 2015).

[49] Moxiecode Systems AB, TinyMCE, http://www.tinymce.com/ (14. 05. 2015).

[50] J. Burke, K. Westin, K. Harsh, M. Medeiros, RequireJs Text Resource Loader Plugin, https://github.com/requirejs/text (14. 05. 2015).

[51] H. Elias, Perlin Noise, http://freespace.virgin.net/hugo.elias/models/m_perlin.htm (14. 05. 2015).

[52] C. Parker, Boids Pseudocode, http://www.kfish.org/boids/pseudocode.html (14. 05. 2015).

[53] A. Gryc, Improvements to the canonical one-liner GLSL rand() for OpenGL ES 2.0, http://byteblacksmith.com/improvements-to-the-canonical-one-liner-glsl-rand-for-opengl-es-2-0/ (14. 05. 2015).

[54] R. Cabello et al, Three.js r43, https://github.com/mrdoob/three.js/releases/tag/r43 (14. 05. 2015).

[55] P. Shirley, M. Ashikhmin, S. Marschner, Fundamentals of Computer Graphics, 2009, A K Peters/CRC Press.

[56] Nvidia, SDK White Paper: Improve Batching Using Texture Atlases, https://developer.nvidia.com/sites/default/files/akamai/tools/files/Texture_Atlas_Whitepaper.pdf (14. 05. 2015).

[57] E. Persson, Tallinn, http://www.humus.name/index.php?page=Textures&ID=101 (14. 05. 2015).

[58] N. Dodgson , Bezier Curves, http://www.cl.cam.ac.uk/teaching/2000/AGraphHCI/SMEG/node3.html (14. 05. 2015).

[59] C.-K. Shene, Derivatives of a Bézier Curve, http://www.cs.mtu.edu/~shene/COURSES/cs3621/NOTES/spline/Bezier/bezier-der.html (14. 05. 2015).

[60] C.-K. Shene, B-spline Basis Functions: Definition, http://www.cs.mtu.edu/~shene/COURSES/cs3621/NOTES/spline/B-spline/bspline-basis.html (14. 05. 2015).

[61] U. Assarsson , Curves and Surfaces, http://www.cse.chalmers.se/edu/year/2011/course/TDA361_Computer_Graphics/Curves%20and%20Surfaces.pdf (14. 05. 2015).

[62] K. Perlin, Making Noise, http://www.noisemachine.com/talk1/index.html (14. 05. 2015).

[63] P. Prusinkiewicz, A. Lindenmayer, The Algorithmic Beauty of Plants, http://algorithmicbotany.org/papers/abop/abop.pdf (14. 05. 2015).

[64] C. Reynolds, Boids, http://www.kfish.org/boids/pseudocode.html (14. 05. 2015).

[65] T. Möller, B. Trumbore, Fast, Minimum Storage Ray/Triangle Intersection, Journal of Graphics Tools, 1997.

[66] R. Tunnel, A. Soikonen, J. Valdma, 2D Nearest Neighbor Search, http://nns.tume-maailm.pri.ee/ (14. 05. 2015).

[67] P. Krishnamachari, Global Illumination in a Nutshell, http://www.thepolygoners.com/tutorials/GIIntro/GIIntro.htm (14. 05. 2015).

[68] F. Boesch, Soft Shadow Mapping, http://codeflow.org/entries/2013/feb/15/soft-shadow-mapping/ (14. 05. 2015).

[69] T. Martin, T.-S. Tan, Anti-aliasing and Continuity with Trapezoidal Shadow Maps, http://www.comp.nus.edu.sg/~tants/tsm.html (14. 05. 2015).

[70] The Code::Blocks team, Code::Blocks, http://www.codeblocks.org/ (14. 05. 2015).

[71] G-Truc Creation, OpenGL Mathematics, http://glm.g-truc.net/0.9.6/index.html (14. 05. 2015).

[72] The GLFW Development Team, GLFW, http://www.glfw.org/ (14. 05. 2015).

[73] M. Bynens, jsPerf — JavaScript performance playground, http://jsperf.com/bresenham-line/3 (14. 05. 2015).

[74] L. Gritz, E. d'Eon, The Importance of Being Linear, http://http.developer.nvidia.com/GPUGems3/gpugems3_ch24.html (14. 05. 2015).

[75] R. Cabello et al, Three.js / examples / animation / skinning / blending, http://threejs.org/examples/#webgl_animation_skinning_blending (14. 05. 2015).

[76] L. Kaplinski, Reflective water with GLSL, Part I, http://khayyam.kaplinski.com/2011/09/reflective-water-with-glsl-part-i.html (14. 05. 2015).

[77] H. Prautzsch, W. Boehm, M. Paluszny, Bézier and B-Spline Techniques, 2002, Springer.

[78] D. L. Finn, MA 323 Geometric Modelling Course Notes: Day 18 Bezier Splines II, https://www.rose-hulman.edu/~finn/CCLI/Notes/day18.pdf (14. 05. 2015).

Appendix

This appendix includes illustrations and tables that were too big to fit inside the thesis. There is also an archive accompanying this thesis that includes:

- Questionnaire (with results)
- Lecture slides (including the ones from other lecturers for archiving purposes)
- Exit cards
- CGLearn's source code (with setup manual and database export)

Table 15: Schedule of lecture and practice session topics.

Week	Event	Conductor	Topic
1	Lecture	Raimond Tunnel	Introduction to Computer Graphics.
	Practice	Raimond Tunnel	Introduction to Allegro, OpenGL, HTML Canvas, WebGL, Three.js.
	Practice	Margus Luik	
2	Lecture	Raimond Tunnel	Geometry and Vectors.
	Practice	Raimond Tunnel	Rasterization.
	Practice	Margus Luik	
3	Lecture	Raimond Tunnel	Transformations, Matrix Stack.
	Practice	Raimond Tunnel	
	Practice	Margus Luik	
4	Lecture	Raimond Tunnel	Frames of Reference, Projection.
	Practice	Raimond Tunnel	
	Practice	Margus Luik	
5	Lecture	Raimond Tunnel	Shading and Lighting.
	Practice	Raimond Tunnel	
	Practice	Margus Luik	
6	Lecture	Raimond Tunnel	Textures and Sampling.
	Practice	Raimond Tunnel	
	Practice	Margus Luik	
7	Lecture	Jaanus Jaggo	Blending.
	Practice		
	Practice		
8	Lecture	Raimond Tunnel	Recapitulation.
	Practice	Timo Kallaste	Introduction to Maya and Blender.
	Practice		
9	Lecture	Ats Kurvet, Timo Kallaste	Modeling and Game Engines.
	Practice	Margus Luik	Model importing.
	Practice	Timo Kallaste	Modeling and Texturing in Blender
10	Lecture	Benson Muite	Data Visualization.
	Practice	Margus Luik	Environment Mapping.
	Practice	Timo Kallaste	Rigging and Animation.

Continues on the next page.

Week	Event	Conductor	Topic
11	Lecture	Raimond Tunnel	Curves.
	Practice		
	Practice	Margus Luik Ats Kurvet	Introduction to Unreal Engine 4 and Unity 3D
12	Lecture	Jaanus Jaggo Raimond Tunnel	Procedural Generation.
	Practice	Jaanus Jaggo	
	Practice	Timo Kallaste Ats Kurvet	Scripting and Blueprint.
13	Lecture	Raimond Tunnel	Ray Tracing, Space Partitioning, BVH.
	Practice	Raimond Tunnel	
	Practice	Timo Kallaste Ats Kurvet	Scripting Materials.
14	Lecture	Raimond Tunnel	Global Illumination.
	Practice		
	Practice	Timo Kallaste Ats Kurvet	Materials, Effects, Animation.
15	Lecture	Raimond Tunnel	Shadows. Conclusion.
	Practice	Margus Luik	
	Practice	Timo Kallaste Ats Kurvet	Lightmass. Static vs Dynamic lighting.
16	Lecture	Raimond Tunnel	Project Presentations.

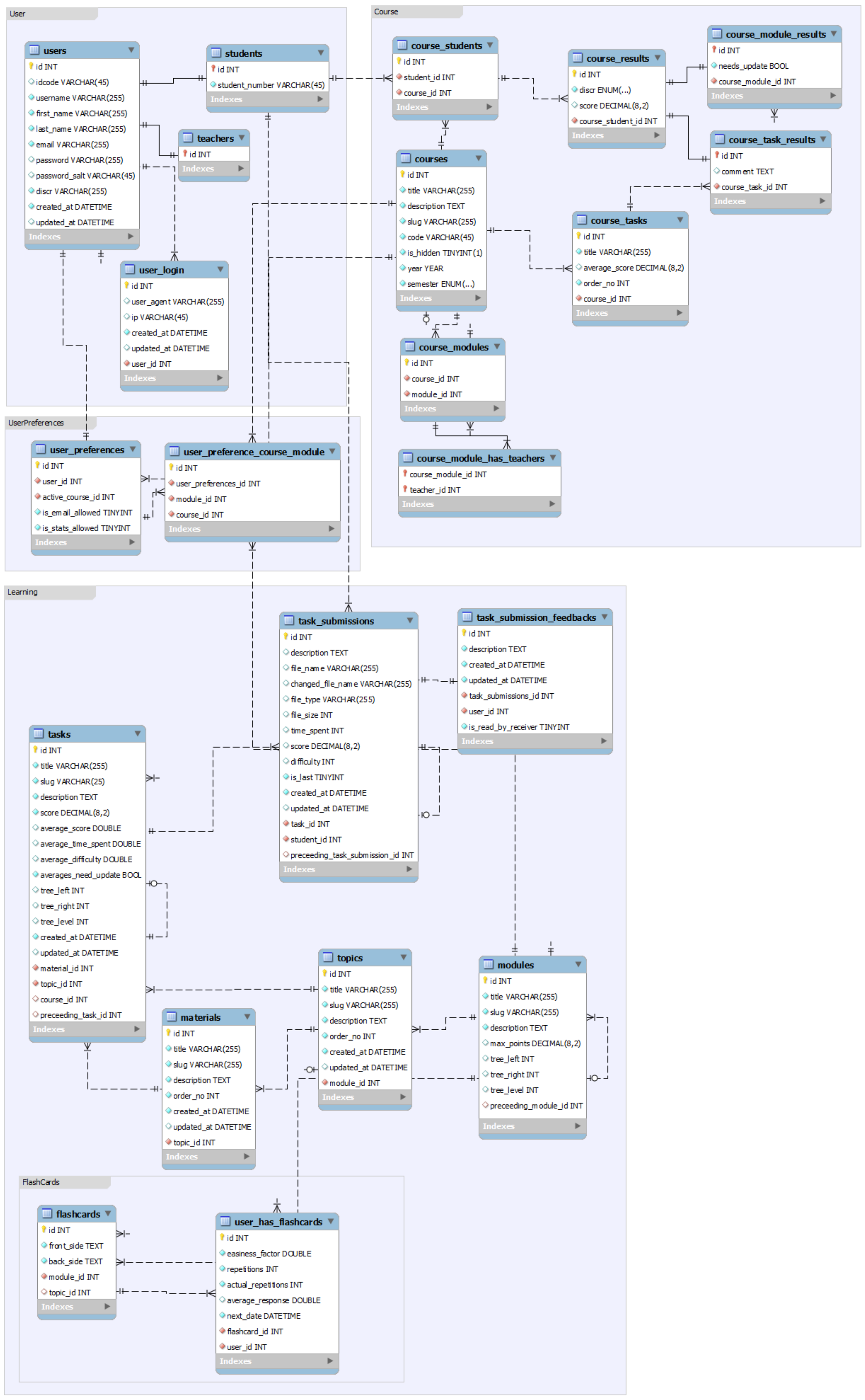
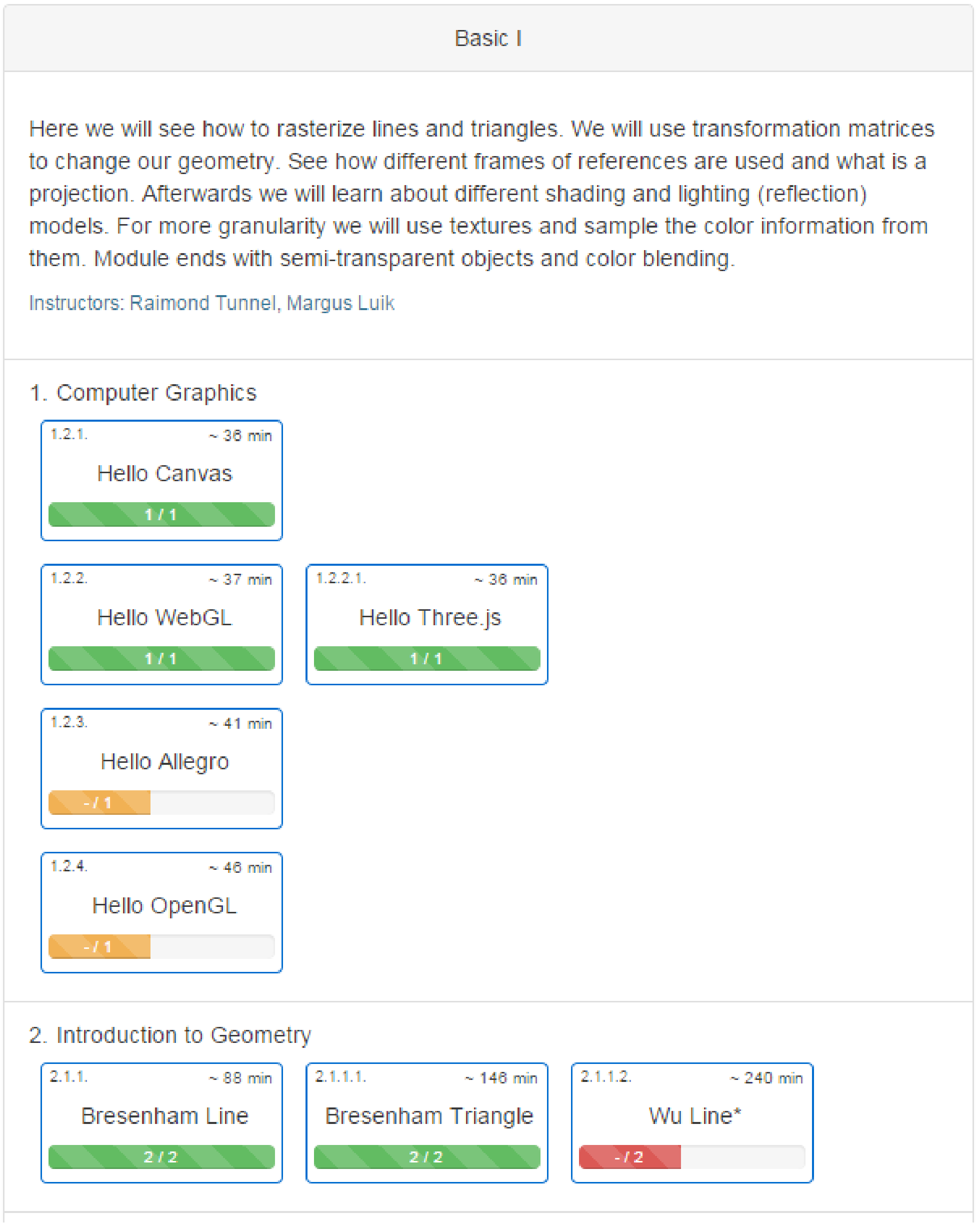


Illustration 84: Database model constructed in MySQL Workbench.


Basic I

Here we will see how to rasterize lines and triangles. We will use transformation matrices to change our geometry. See how different frames of references are used and what is a projection. Afterwards we will learn about different shading and lighting (reflection) models. For more granularity we will use textures and sample the color information from them. Module ends with semi-transparent objects and color blending.

Instructors: Raimond Tunnel, Margus Luik

1. Computer Graphics

1.2.1. ~ 36 min Hello Canvas 1 / 1

1.2.2. ~ 37 min Hello WebGL 1 / 1

1.2.2.1. ~ 36 min Hello Three.js 1 / 1

1.2.3. ~ 41 min Hello Allegro - / 1

1.2.4. ~ 46 min Hello OpenGL - / 1

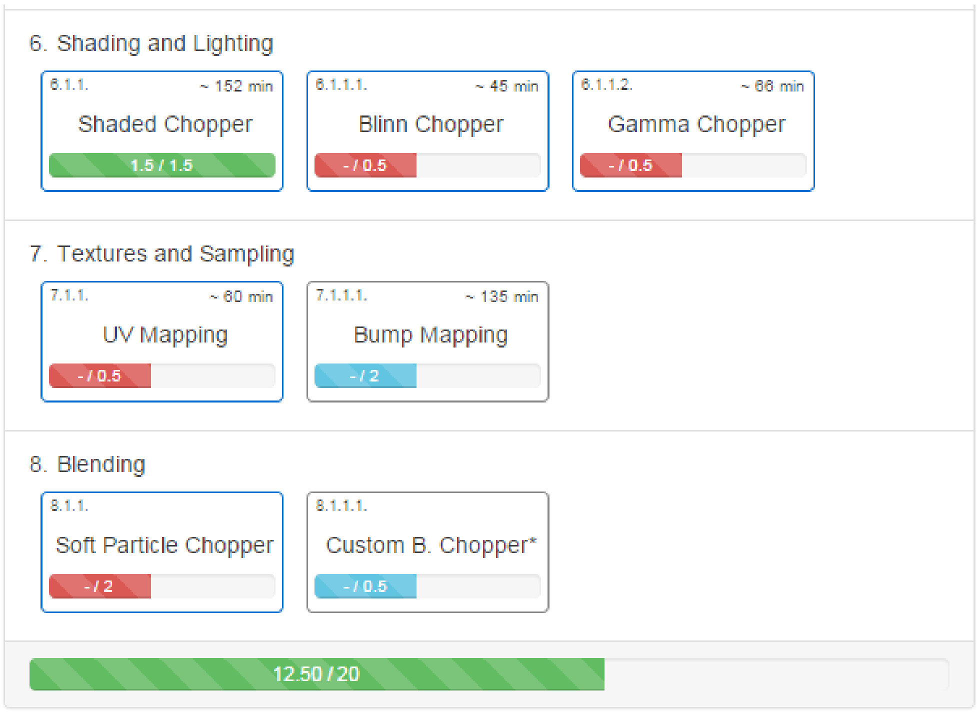
2. Introduction to Geometry

2.1.1. ~ 88 min Bresenham Line 2 / 2

2.1.1.1. ~ 146 min Bresenham Triangle 2 / 2

2.1.1.2. ~ 240 min Wu Line* - / 2

Illustration 85: Part of the tasks tree. Basic I module is opened. Student has submitted all the tasks for the first week. The tasks Hello Allegro and Hello OpenGL have not yet been graded by an instructor. Student has also not submitted a solution for the Wu Line task.



6. Shading and Lighting

6.1.1. ~ 152 min Shaded Chopper 1.5 / 1.5

6.1.1.1. ~ 45 min Blinn Chopper - / 0.5

6.1.1.2. ~ 68 min Gamma Chopper - / 0.5

7. Textures and Sampling

7.1.1. ~ 60 min UV Mapping - / 0.5

7.1.1.1. ~ 135 min Bump Mapping - / 2

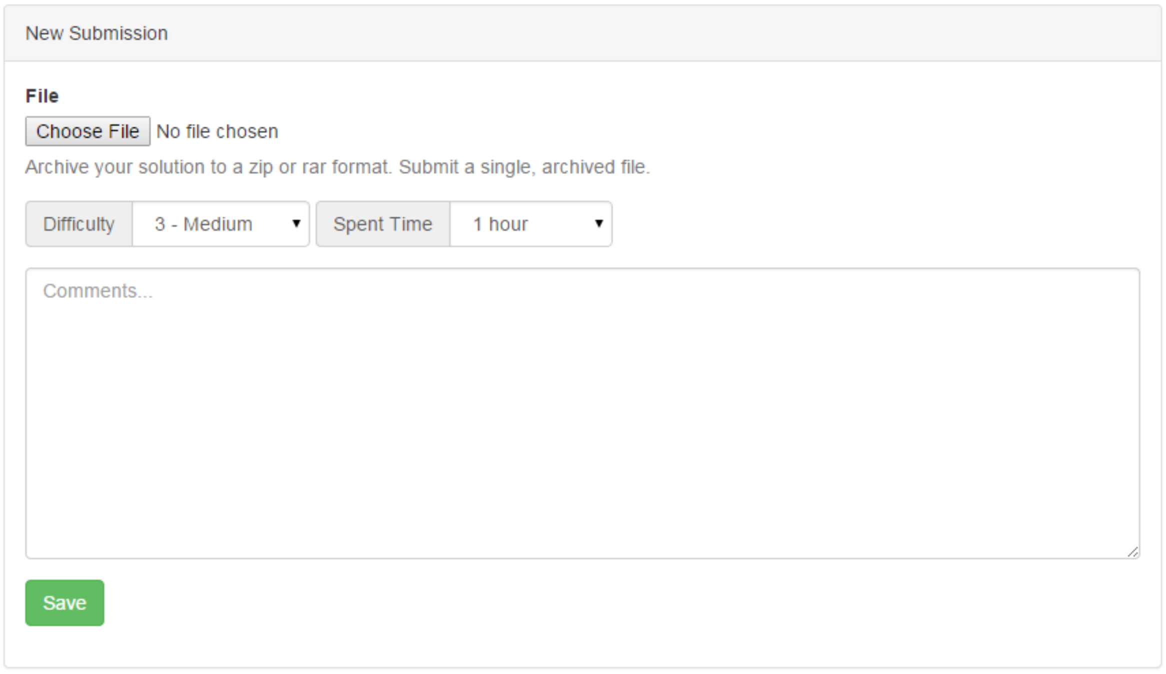
8. Blending

8.1.1. Soft Particle Chopper - / 2

8.1.1.1. Custom B. Chopper* - / 0.5

12.50 / 20

Illustration 86: Part of the Basic I tasks tree, a continuation of the previous illustration. Here it is visible that the student has to solve UV Mapping task before the Bump Mapping task's solution can be submitted. Similarl case with the Soft Particle Chopper and Custom B. Chopper tasks. The latter has a prerequisite that the former should have a solution submitted to. At the end of the module, a total score earned by this student from this module is shown.*



New Submission

File

Choose File No file chosen

Archive your solution to a zip or rar format. Submit a single, archived file.

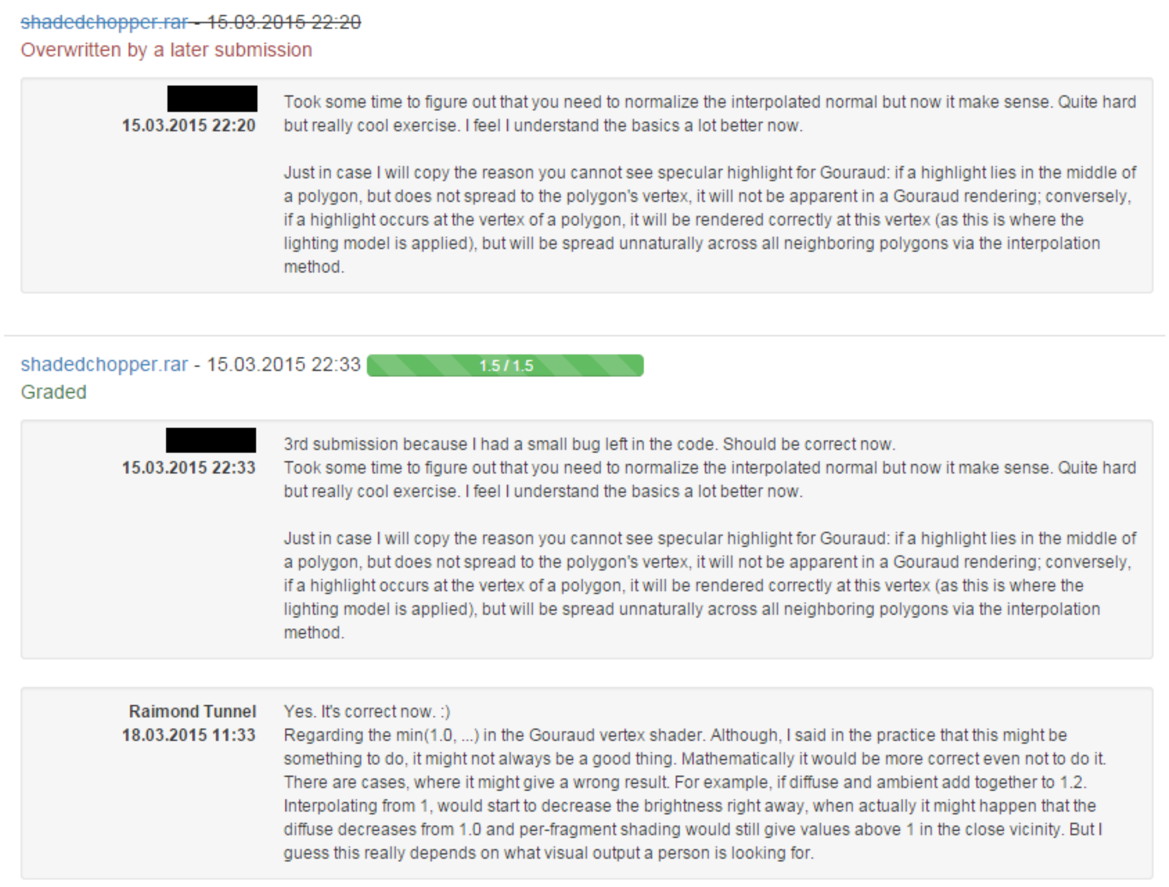
Difficulty: 3 - Medium

Spent Time: 1 hour

Comments...

Save

Illustration 87: Submission form for a task. Besides uploading a file and writing comments about the solution, this form asks the student to assess the task's difficulty and spent time.


~~shadedchopper.rar - 15.03.2015 22:20~~
Overwritten by a later submission

[redacted]
15.03.2015 22:20
Took some time to figure out that you need to normalize the interpolated normal but now it make sense. Quite hard but really cool exercise. I feel I understand the basics a lot better now.

Just in case I will copy the reason you cannot see specular highlight for Gouraud: if a highlight lies in the middle of a polygon, but does not spread to the polygon's vertex, it will not be apparent in a Gouraud rendering; conversely, if a highlight occurs at the vertex of a polygon, it will be rendered correctly at this vertex (as this is where the lighting model is applied), but will be spread unnaturally across all neighboring polygons via the interpolation method.

shadedchopper.rar - 15.03.2015 22:33 1.5 / 1.5
Graded

[redacted]
15.03.2015 22:33
3rd submission because I had a small bug left in the code. Should be correct now.
Took some time to figure out that you need to normalize the interpolated normal but now it make sense. Quite hard but really cool exercise. I feel I understand the basics a lot better now.

Just in case I will copy the reason you cannot see specular highlight for Gouraud: if a highlight lies in the middle of a polygon, but does not spread to the polygon's vertex, it will not be apparent in a Gouraud rendering; conversely, if a highlight occurs at the vertex of a polygon, it will be rendered correctly at this vertex (as this is where the lighting model is applied), but will be spread unnaturally across all neighboring polygons via the interpolation method.

Raimond Tunnel
18.03.2015 11:33
Yes. It's correct now. :)
Regarding the min(1.0, ...) in the Gouraud vertex shader. Although, I said in the practice that this might be something to do, it might not always be a good thing. Mathematically it would be more correct even not to do it. There are cases, where it might give a wrong result. For example, if diffuse and ambient add together to 1.2. Interpolating from 1, would start to decrease the brightness right away, when actually it might happen that the diffuse decreases from 1.0 and per-fragment shading would still give values above 1 in the close vicinity. But I guess this really depends on what visual output a person is looking for.

Illustration 88: Multiple submissions with corrections submitted for the same task. A discussion between the student and the instructor is shown.

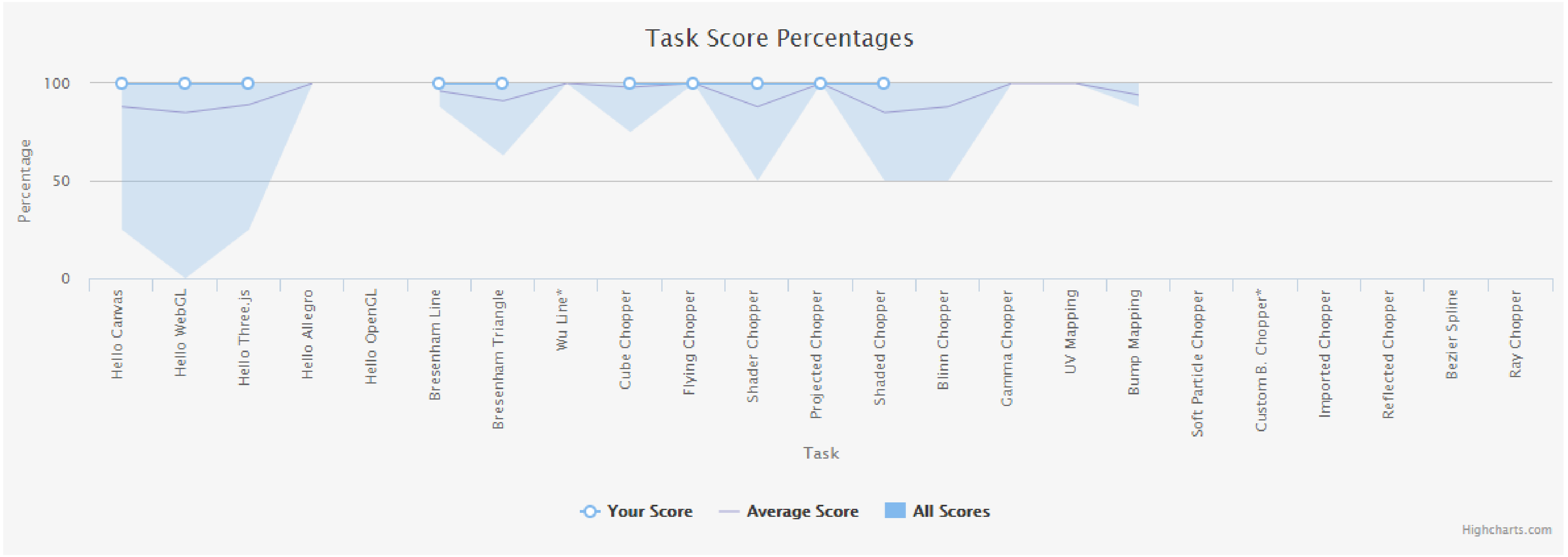


Illustration 89: Statistics for the task scores.

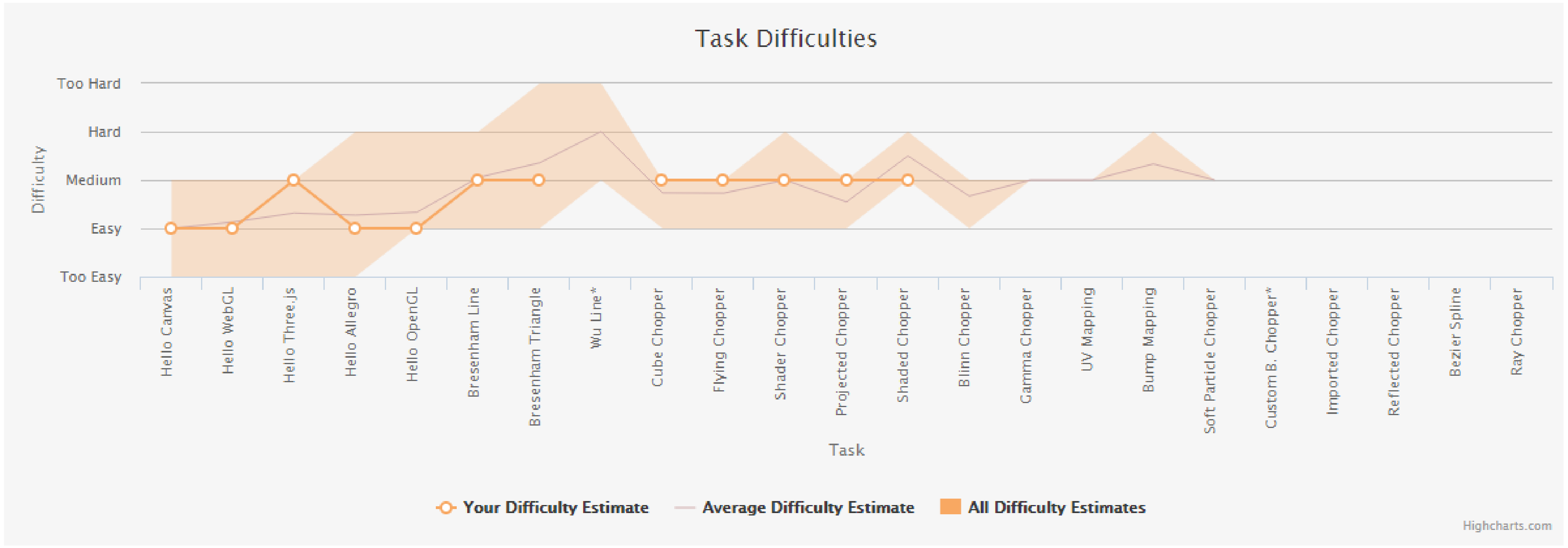


Illustration 90: Statistics for the task difficulties.

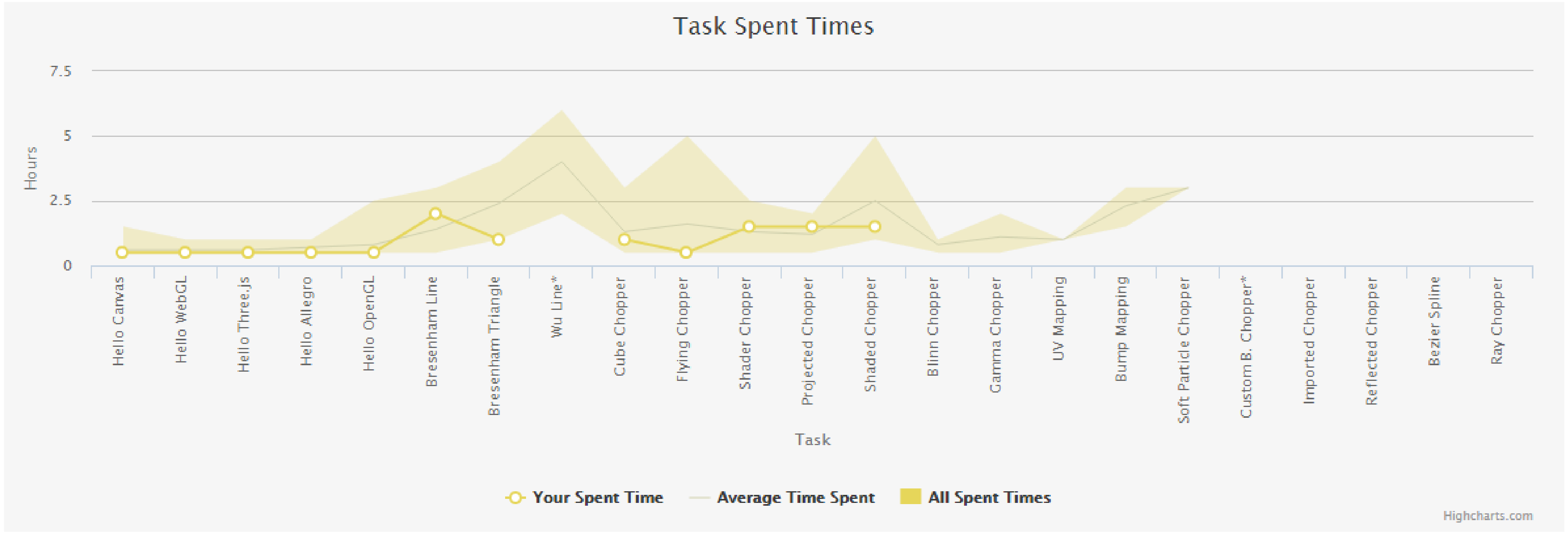
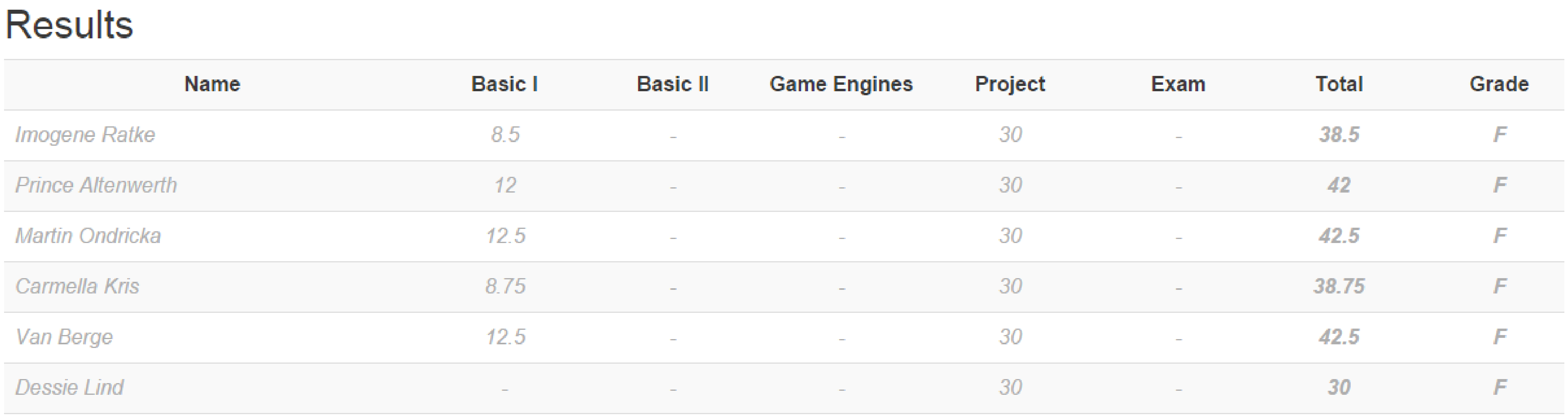


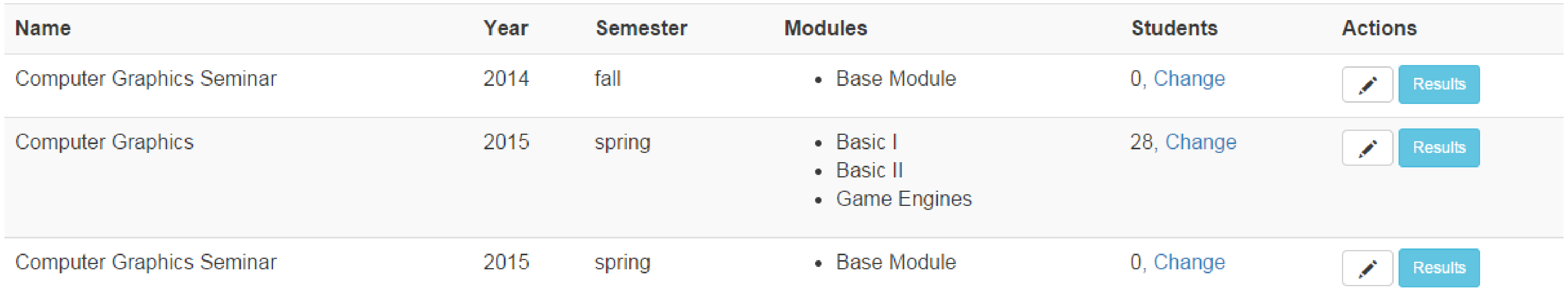
Illustration 91: Statistics for the task time estimates.



Results

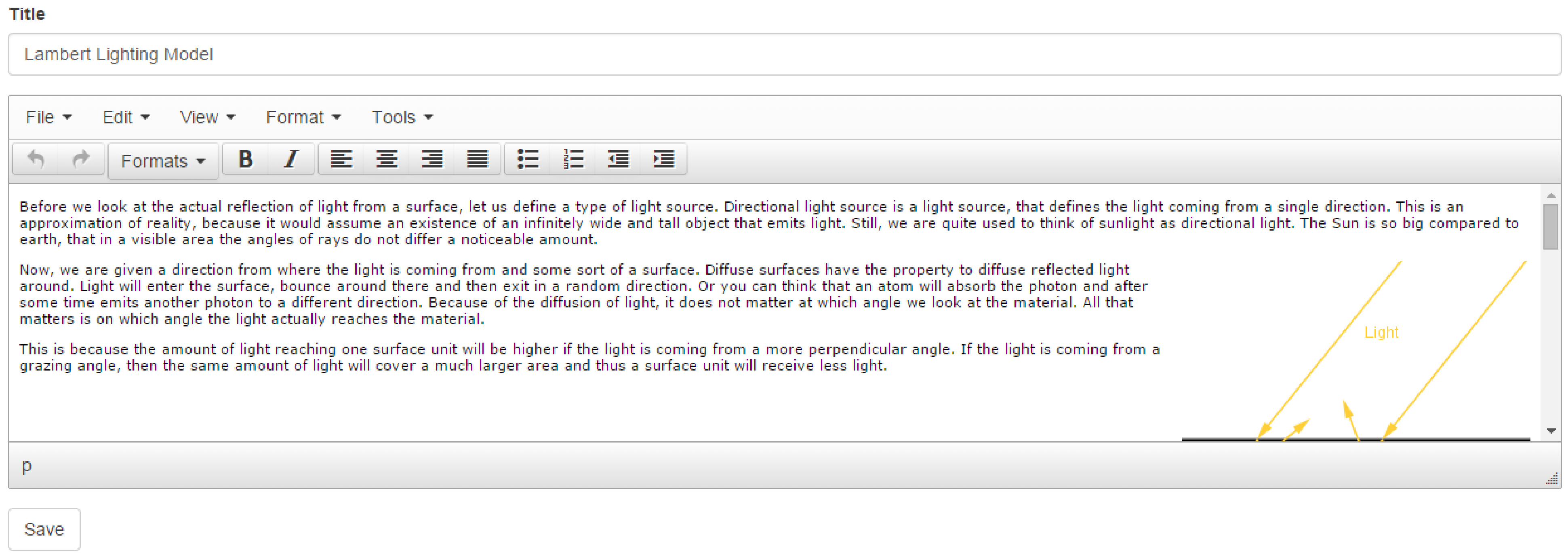
Name	Basic I	Basic II	Game Engines	Project	Exam	Total	Grade
Imogene Ratke	*8.5*	-	-	*30*	-	***38.5***	***F***
Prince Altenwerth	*12*	-	-	*30*	-	***42***	***F***
Martin Ondricka	*12.5*	-	-	*30*	-	***42.5***	***F***
Carmella Kris	*8.75*	-	-	*30*	-	***38.75***	***F***
Van Berge	*12.5*	-	-	*30*	-	***42.5***	***F***
Dessie Lind	-	-	-	*30*	-	***30***	***F***

Illustration 92: Part of the results table. Other students have grayed rows and fake names.



Name	Year	Semester	Modules	Students	Actions
Computer Graphics Seminar	2014	fall	• Base Module	0, Change	Results
Computer Graphics	2015	spring	• Basic I • Basic II • Game Engines	28, Change	Results
Computer Graphics Seminar	2015	spring	• Base Module	0, Change	Results

Illustration 93: Course list. The Computer Graphics Seminar does not have any tasks, so the students do not have to be assigned to it in the system. The button with a pencil icon opens the editing view of the course. This is similar for other lists aswell.



Lambert Lighting Model

Before we look at the actual reflection of light from a surface, let us define a type of light source. Directional light source is a light source, that defines the light coming from a single direction. This is an approximation of reality, because it would assume an existence of an infinitely wide and tall object that emits light. Still, we are quite used to think of sunlight as directional light. The Sun is so big compared to earth, that in a visible area the angles of rays do not differ a noticeable amount.

Now, we are given a direction from where the light is coming from and some sort of a surface. Diffuse surfaces have the property to diffuse reflected light around. Light will enter the surface, bounce around there and then exit in a random direction. Or you can think that an atom will absorb the photon and after some time emits another photon to a different direction. Because of the diffusion of light, it does not matter at which angle we look at the material. All that matters is on which angle the light actually reaches the material.

This is because the amount of light reaching one surface unit will be higher if the light is coming from a more perpendicular angle. If the light is coming from a grazing angle, then the same amount of light will cover a much larger area and thus a surface unit will receive less light.

Illustration 94: Editing a material with TinyMCE.

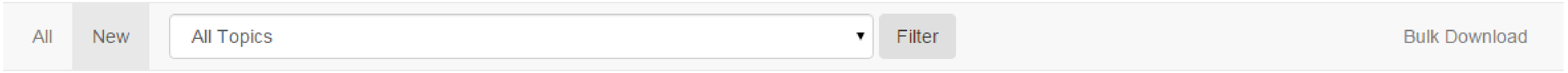
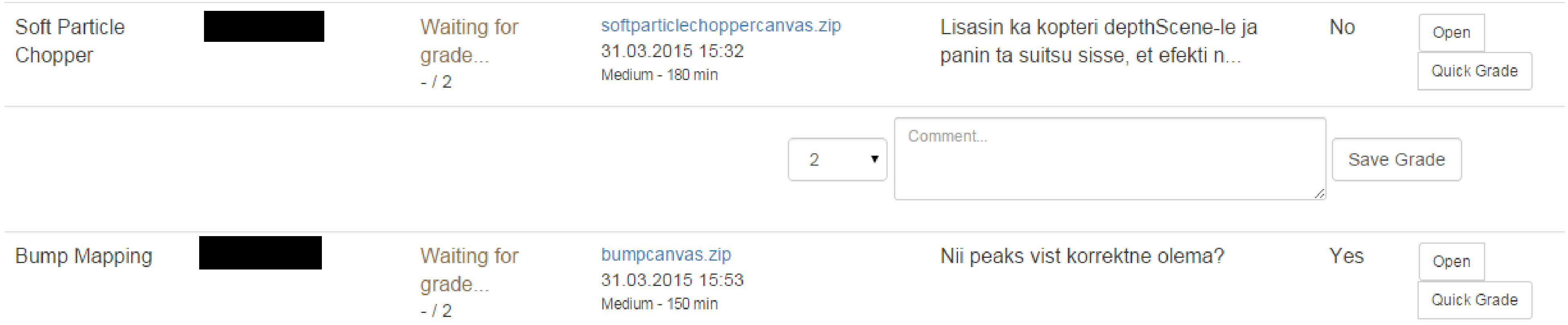
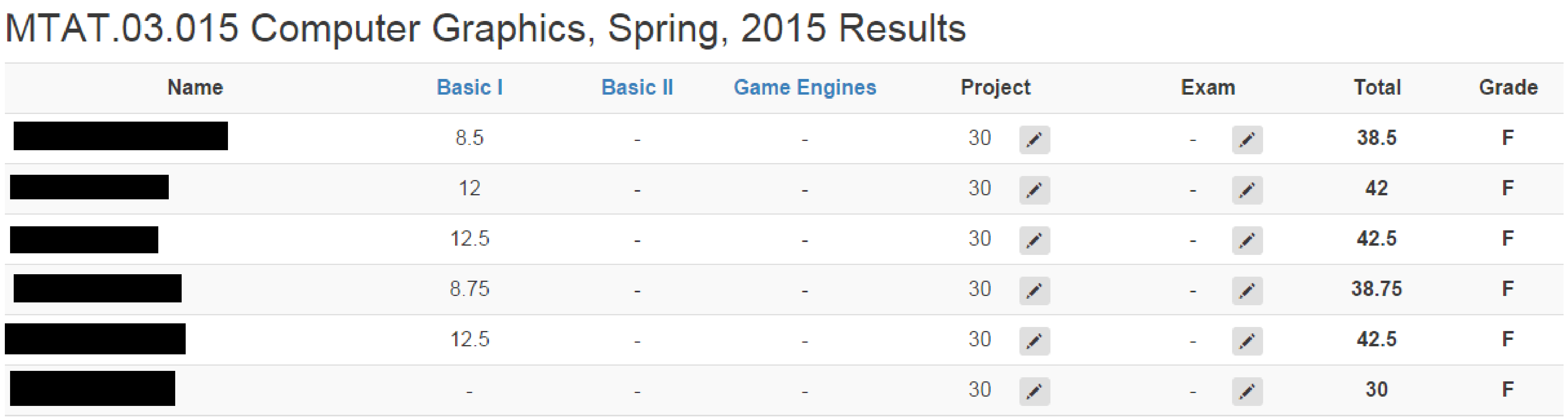


Illustration 95: The top menu of the Submissions page. It allows filtering the new submissions via a certain topic. There is also a button for the Bulk Download that downloads all the new submissions packed with a well-organized file structure.



Soft Particle Chopper	[redacted]	Waiting for grade... - / 2	softparticlechoppercanvas.zip 31.03.2015 15:32 Medium - 180 min	Lisasin ka kopteri depthScene-le ja panin ta suitsu sisse, et efekti n...	No	Open Quick Grade
			2	Comment...	Save Grade	
Bump Mapping	[redacted]	Waiting for grade... - / 2	bumpcanvas.zip 31.03.2015 15:53 Medium - 150 min	Nii peaks vist korrektne olema?	Yes	Open Quick Grade

Illustration 96: Two submitted solutions in the submissions table. The first one has the Quick Grade form opened. When selecting the correct score, writing the comment and clicking on the Save Grade button, the form will close and the teacher can continue with the next solution.



MTAT.03.015 Computer Graphics, Spring, 2015 Results

Name	Basic I	Basic II	Game Engines	Project	Exam	Total	Grade
[redacted]	8.5	-	-	30	-	**38.5**	**F**
[redacted]	12	-	-	30	-	**42**	**F**
[redacted]	12.5	-	-	30	-	**42.5**	**F**
[redacted]	8.75	-	-	30	-	**38.75**	**F**
[redacted]	12.5	-	-	30	-	**42.5**	**F**
[redacted]	-	-	-	30	-	**30**	**F**

Illustration 97: Results table in the teacher's module. The points for the project and the exam can be edited.

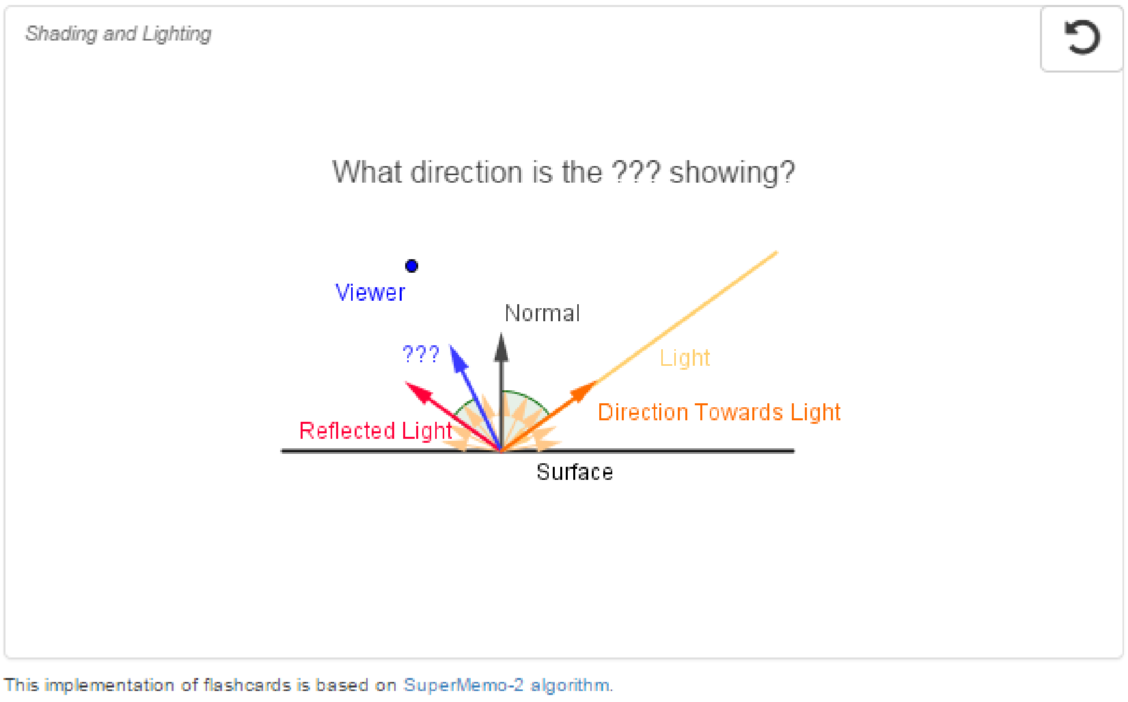

Shading and Lighting
What direction is the ??? showing?
Viewer
Normal
???
Light
Direction Towards Light
Reflected Light
Surface
This implementation of flashcards is based on SuperMemo-2 algorithm.

Illustration 98: Example of the question side of a flashcard.

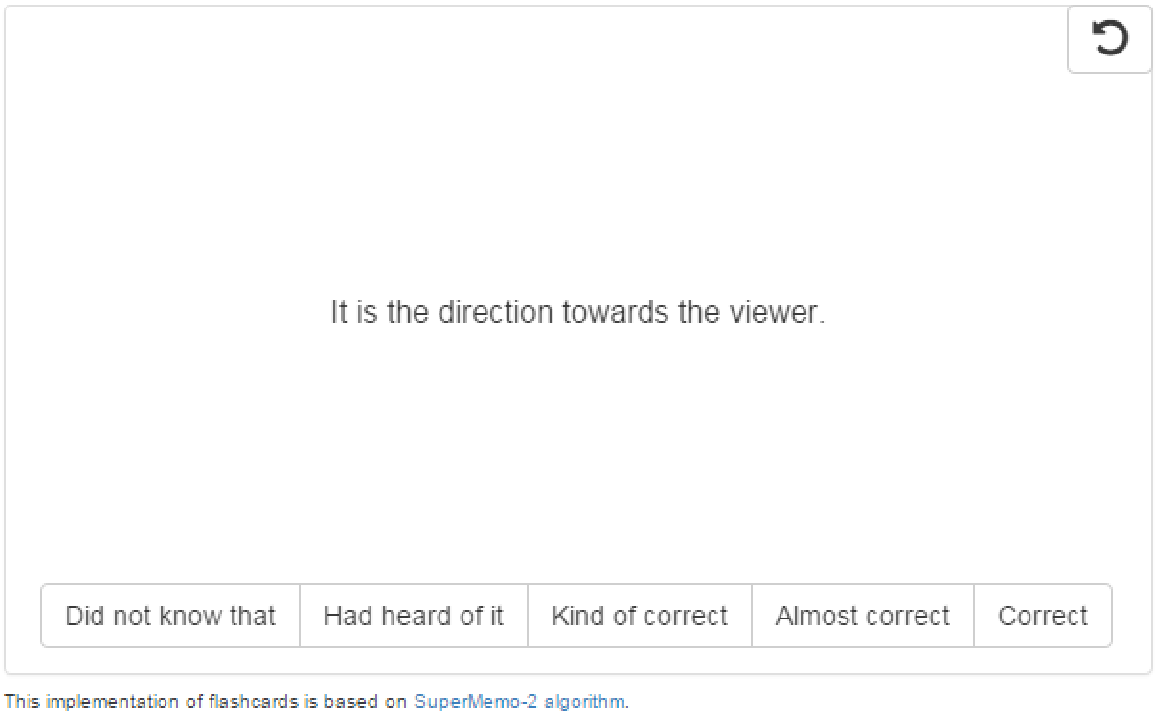

It is the direction towards the viewer.
Did not know that
Had heard of it
Kind of correct
Almost correct
Correct
This implementation of flashcards is based on SuperMemo-2 algorithm.

Illustration 99: Example of the answer side of the flashcard. Buttons allow the student to assess their previously considered answer.

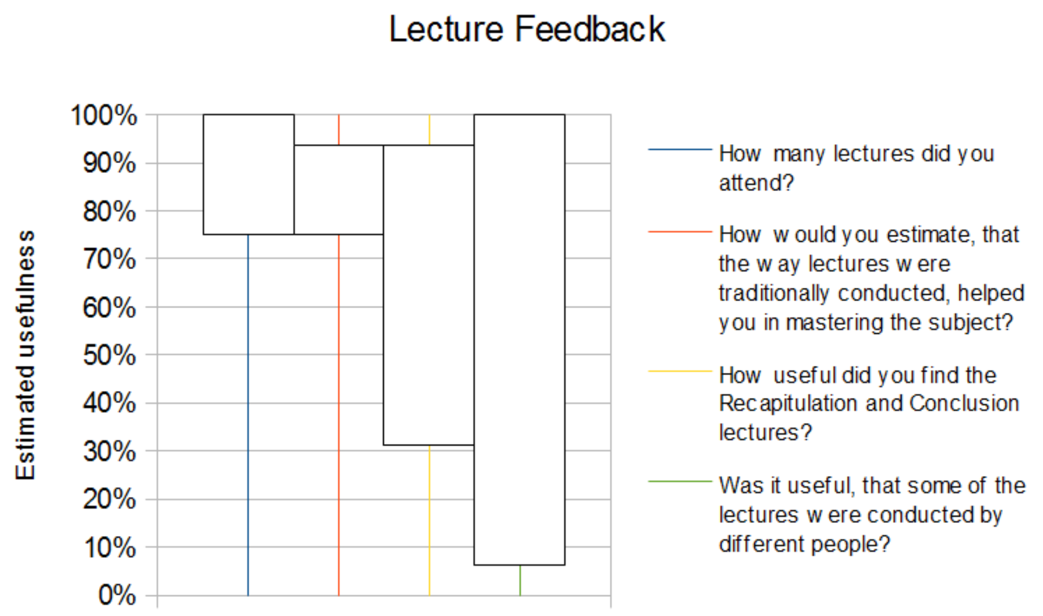


Illustration 100: Box plots reflecting the answers for the lecture feedback.

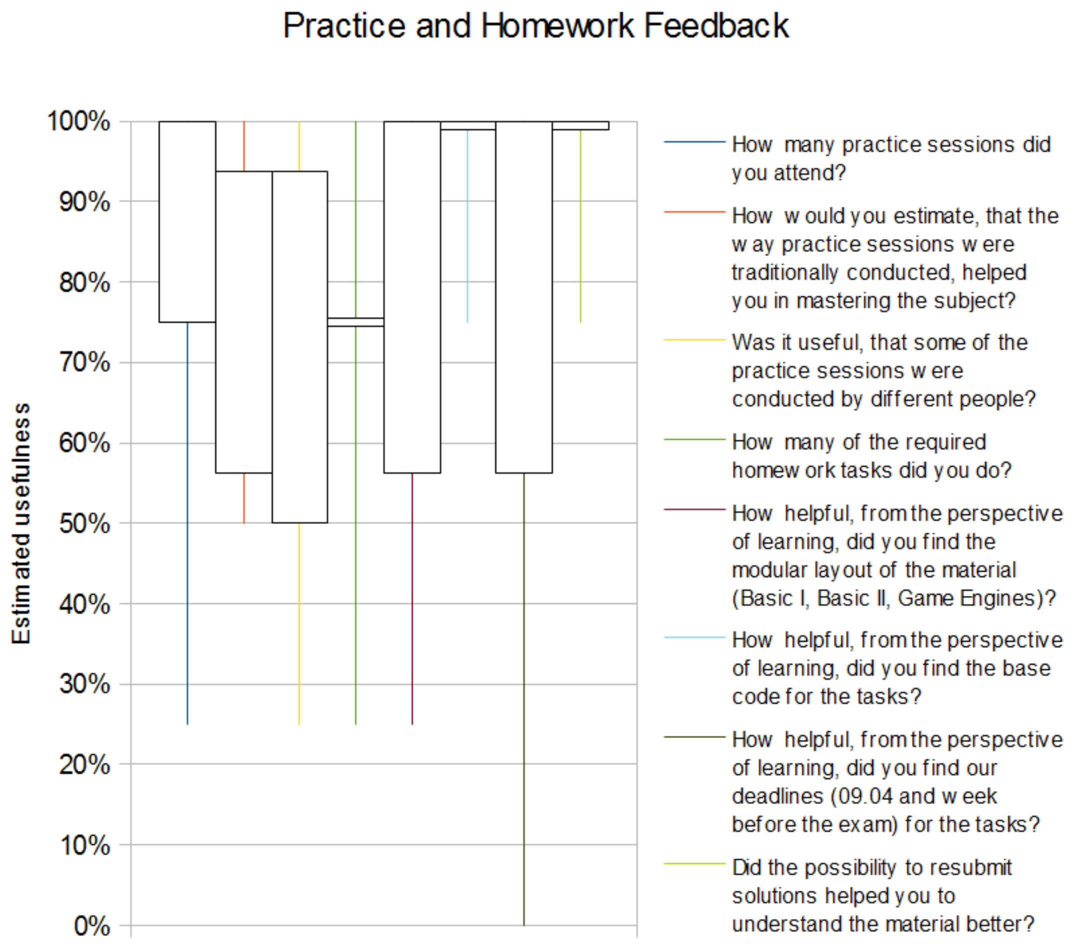


Illustration 101: Box plots reflecting the answers for the practice sessions and homework feedback.

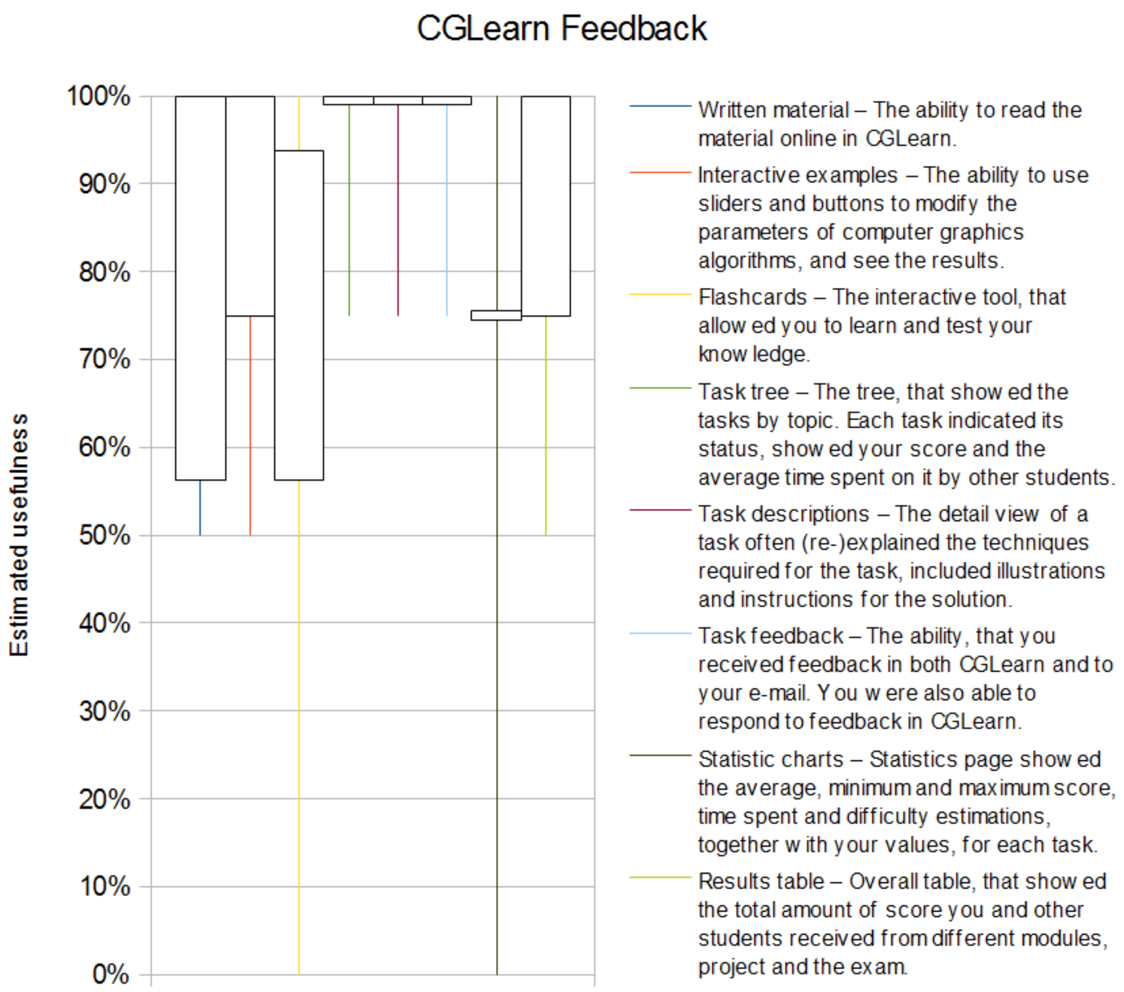


Illustration 102: Box plots reflecting the answers for the CGLearn functionality feedback.