How to Increase Interest in Studying Functional Programming via Interdisciplinary Application

Pedro Figueirêdo Yuri Kim Nghia Le Minh Evan Sitt Xue Ying
Viktória Zsók

Eötvös Loránd University, Faculty of Informatics
Department of Programming Languages and Compilers
Budapest, Hungary

pedrofigueiredo5206@gmail.com, kyr9412@gmail.com, minhnghia.1999@gmail.com
Sitt.Evan@gmail.com, xueying19981206@gmail.com, zsv@inf.elte.hu

Functional programming represents a modern tool for applying and implementing software. The state of the art in functional programming reports an increasing number of methodologies in this paradigm. However, extensive interdisciplinary applications are missing. Our goal is to increase student interest in pursuing further studies in functional programming with the use of an application: the ray tracer. We conducted a teaching experience, with positive results and student feedback, described here in this paper.

1 Introduction

In recent years, functional programming has become one of the first major programming language courses [2, 7, 11]. This paper reports the results and the outcomes of an application oriented teaching experience to make a functional programming course more attractive and motivating for first year, first semester students. The experiment was conducted over the span of two semesters from two different academic years. Our goal is to fill the gap between a theoretical teaching approach and application oriented software labs.

Throughout our classes, large amounts of simple exercises were provided to the students so that they could understand recursion, guarded expressions, parameters, and other major functional programming concepts. Past experience has shown that these exercises were helping them; however, they were also keen on applications where they could see functional programming in real world scenarios.

As students come from vastly different backgrounds in high school mathematics and physics, they often struggle to understand the basics and can get lost at the initial stages. This causes the students to become unable to progress in later classes, since the curriculum is fast-paced and iterative.

We considered presenting an especially attractive computer graphics application - ray tracing - wherein, after short introductory presentations, they can extend a partially complete Abstract Data Type (ADT) with new functions and try out well-known concepts, utilized in computer graphics, engineering, medicine, and data visualization applications.

Ray tracing usually is studied extensively in the framework of more advanced courses. However, here we provided first a simple background, then the implementation for the complex parts, leaving it up to the students to extend it by the minor missing properties and operations required to visualize graphical elements. Hence, students were engaged and able to practice various functional programming concepts in the scope of the application and perceive how the acquired language elements would work in practice.

J. Hage (Ed.): Trends in Functional Programming
in Education 2019 and 2020 (TFPIE '19 and '20)
EPTCS 321, 2020, pp. 37–54, doi:10.4204/EPTCS.321.3

In this paper, we describe our functional programming course structure (section 2), the motivation of the students via the ray tracing application (section 3), the integration of the application into the course (section 4), the presentation and discussion of our results (section 5), and lastly future work (section 6) with conclusions (section 7).

2 Teaching Introduction to Functional Programming

At our university, Eötvös Loránd University, an introductory course to functional programming is offered as a compulsory part of the first year first semester curriculum for Computer Science bachelor's students. This is taught concurrently alongside imperative and procedural paradigm courses, such as C/C++ and Python.

In this section, the class structure of our functional programming course (2.1) and the challenges of teaching functional programming to newly admitted students (2.2) are discussed.

2.1 Functional Programming Class Structure

The Functional Programming course is divided into two components. There is a two-hour lecture component and a two-hour practical component. While the lecture component is composed of presentations focused on the theoretical part, the practical component relies on implementation techniques as its primary method of delivering information. The teacher to student ratio is 3:66. In the years prior to our experiment, there was only a two-hour practical session which severely limited the time available, with a teacher to student ratio of 2:83.

There are ten lecture sessions for the course, one per week during the duration of the semester with one week of vacation in the middle of the semester. Each of these sessions focuses on a different aspect of functional programming. They are structured to form a step-wise progression for first-year, first-semester students with no programming knowledge to learn the basics of the functional programming paradigm all the way up to basic abstractions.

Within the span of the semester, we cover many basic fundamental concepts of functional programming. We start with types, recursion, and function comprehension. Afterwards the course will proceed to lists and list comprehensions, tuples, and records and instances. After the midterm, the lecture proceeds to discuss arrays, abstract data types, and trees. And finally, the course wraps up with classes and input/output.

As the students are first-year, first-semester students, the students are not expected to, nor do most possess, any preliminary knowledge or experience in programming. This means that the lectures not only include functional programming concepts, but must also include general programming concepts and algorithms.

For each lecture session there is a mandatory practical session within the same week. During these practical sessions, the students are given classwork exercises formulated around the theoretical concepts taught during the lecture session. The instructors for the practical sessions help guide the students through these exercises in Clean [1]. Instructors show how to solve some of those exercises explaining the underlying algorithms and logic which students can use to solve further examples. Additionally, the students are given a short theoretical quiz that helps reinforcing the theoretical concepts that were presented during the prior lecture session along with evaluating their ability to analyze and comprehend code snippets.

In addition to the compulsory lecture and practical sessions each week, a consultation session is offered three times per week. During the consultation sessions, teaching assistants are available for the students to consult with. Students are highly encouraged to take advantage of this resource for their homework and general questions. As the student to instructor ratio is lower during the consultation sessions, the teaching assistants are able to provide more individualized help and explanations to assist each student with understanding the concepts and completing their assignments. Additionally, during the consultations, teaching assistants also help those who are progressing well within the class to learn new methods and techniques in functional programming, along with emphasizing good programming style.

Furthermore, one of our teaching assistants also conducts a more specialized type of consultation via online streaming. These consultations take advantage of current trends in digital media consumption and are held over YouTube. Additional tutorial videos are also made and uploaded to the same channel. This has many advantages over the traditional consultation.

Generally, students were more relaxed as they could attend from the comfort of their dormitory. They had an easier time following the exercises being presented by the teaching assistant as it was on their own screen. Questions were also easier to pose as they could be sent to the chat without disrupting the teaching assistant or drawing attention, and could be addressed by the teaching assistant in an appropriate time. Lastly, with the nature of a digital format, the solved exercises and the recording of the live stream were both easily accessible at any time after the stream, allowing for the students to re-attend the consultation at a later time at their own convenience and allow for students who could not make the scheduled time to have access to it as well.

Each week, the students are assigned a compulsory homework assignment. Each homework assignment has fewer exercises than the classwork assignments, however each exercise is much more sophisticated than the basic classwork exercises. The homework exercises each emphasize and promote a solid understanding of functional programming fundamentals along with the skill for applying and combining them in a close-to-real-life application. Additionally, the difficulty level of the homework is intended to encourage the student to take their time over the span of the week to research and practice the skills required to complete the exercises. It also aims at preparing them for the examination exercises.

Over the course of the semester, students are given two examinations. One is a midterm examination focusing on the first half of the material, while the other is an end term focusing on the second half of the material. Typically the exam questions are selected considering a division regarding difficulties. We try to have basic, intermediate and advanced problems with ratio of 1:2:1. Once instructors and teaching assistants create sample exam questions, they vote what should be included or discarded based on the difficulty criterion and relevance to the course. Students are expected to have a strong understanding of fundamental concepts and the practiced skill to apply them to various applications, and the examination exercises are written with this focus in mind. In the past, it has been observed that the midterm examination also serves as a good measure of progress for the students, and those who take to heart the need for improvement and come to the consultations usually focus on improving on their weaknesses as shown by the midterm.

By the end of the course, the students are expected to be capable of solving application based tasks via the use of data structures and functional programming techniques. They should also be capable of analyzing code written by themselves or others and understand how to debug or optimize it. Lastly, they are encouraged to develop the skills to translate imperative algorithms into optimized functional algorithms.

2.2 Challenges

As our students are first year first semester computer science students, there are considerable challenges that we have faced in teaching them. Amongst these challenges are their lack of experience with programming and their varying level of mathematical proficiency.

The largest of the challenges is introducing the students to concepts of functional programming concurrently with general concepts of programming. As part of the curriculum provided to the instructors, the students learn the imperative paradigm at the same time, and it becomes increasingly difficult to convey concepts such as immutability. Additionally, as the students learn a total of six programming languages within the span of their first semester, the differences between the other languages, all imperative, and Clean add to the challenge.

The varying levels of mathematical proficiency as a result of different educational backgrounds also adds an additional challenge, as the concepts of functional programming are closely tied to mathematical thinking and theory.

Lastly, but certainly not the least, the majority of the students have not found their "true calling" or passion within computer science yet. As such, the majority are studying the material with no motivation, only doing so because they are required to. It is quite a task to emphasize why they should put effort into learning a paradigm that is very different from imperative programming.

Given that prior to our experiment, the course had only one two-hour practical session per week, the time allotted to the instructor was too limited to tackle these challenges. With consideration to the repeated requests of the students for an application for functional programming, we decided to find an application to tackle these challenges.

3 Motivation via Ray Tracing Application

As discussed in Section 2.2, it can be challenging to teach functional programming to first year, first semester students. Therefore, finding an application to demonstrate a use-case of the language could help motivate students to overcome the aforementioned challenges. This section explains the criteria used for selecting an application (3.1), provides a background for ray tracing (3.2), and describes the ray tracing application (3.3) proposed by this paper.

3.1 Criteria for Application Selection

To select an application for motivating students to learn functional programming, it is necessary to decide what constitutes an admissible application for such a goal. For this paper, an admissible application has to fulfill the following requirements:

- interdisciplinary,
- produces concrete output,
- applicable to everyday life of students,
- extensible for further improvements,
- contains basic and advanced levels of functional programming.

An application is considered interdisciplinary if it involves functional programming concepts to solve problems typically encountered in other fields of study. By applying concepts derived from other fields, students can use functional programming to exercise theoretical concepts that are not easily visualized

otherwise. Furthermore, the application produces concrete output, permitting the students to see a result (e.g. an image or technical file) by which they can verify the correctness of their solution to the assignment. This provides assistance to students, who are inexperienced with working on larger projects, bolstering their ability to properly analyse their results.

Moreover, an application is applicable to the everyday life of the majority of the students if they can find inspiration from such interactions and realize real-world uses for their work. An extensible application has the ability of adapting to feedback from students as well as adding new features when needed. Finally, to make use of the application throughout the whole semester of teaching, an admissible application should contain basic and advanced concepts to be tested at multiple occasions during the entirety of the teaching period.

3.2 Background of Ray Tracing

Ray tracing is a technique for image synthesis: creating a 2-D picture of a 3-D world [3]. This rendering technique traces the path of light, converting it to pixels in an image plane, as a way of simulating the effects of multiple light rays interacting with primitives in a virtual environment (Figure 1). Its inherently interdisciplinary background, covering multiple topics spanning both Mathematics and Physics, has shown to be a motivational application of theoretical concepts which are considered hard to understand according to undergraduate students [9].

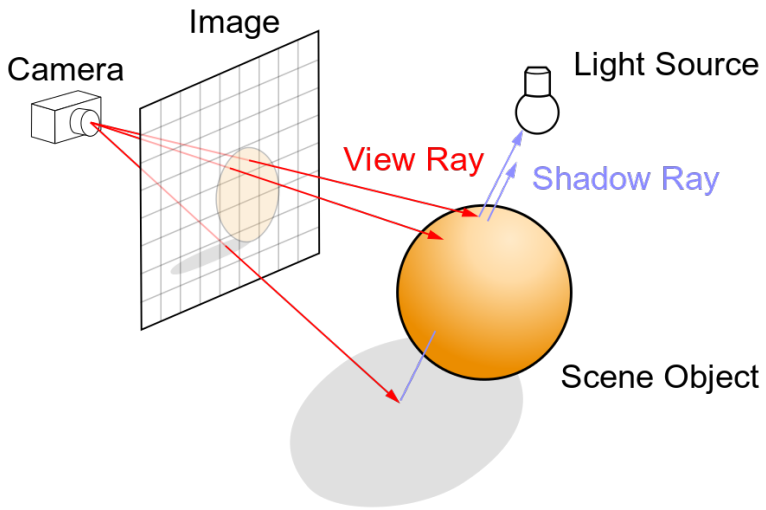

Figure 1: Ray tracing diagram

Implementing this algorithm in the context of education in functional programming courses can also demonstrate how well languages in such a paradigm could handle parallel programming [6, 8, 10]. With ray tracing, considered to be an "embarrassingly parallel" [9] problem, the implementation of such features as well as the understanding of students would be facilitated.

In addition, ray tracing realization encompasses numerous sub-problems, varying in size and complexity, being applicable in different stages of the teaching semester. Beginners could tackle simple algebraic operations while more experienced students can solve ray-triangle intersection tests, for instance. This idea can be applied not only to classwork and homework, but also on tests, thus integrating once minor unconnected tasks into a larger project with a clear and concrete objective: generating images as a result of the ray tracing rendering.

Even though the ray tracing technique is based on various theoretical and lengthy concepts, the core software does not comprise a large project and it does not demand an extensive time investment to be able to display a result. This allows its application by a smaller number of teaching staff. However, the project is not limited to its initial size, being easily expandable and iterative, which can also benefit from a larger staff and experiences from successive semesters.

As it is currently used within a multitude of fields such as engineering, medicine, and data visualization, ray tracing [9] can have a motivational appeal to students. Realizing that, by learning functional programming concepts, one can be a part of a ray tracer software project brings meaningful examples of its importance. Having stated that, ray tracing fulfills all the aforementioned criteria (Section 3.1) for a successful motivating application and was chosen as the motivating application of this paper.

3.3 Ray Tracing Application Description

The ray tracing application is a brute-force simplistic ray tracer implemented entirely in the Clean programming language. It reads triangle meshes from files and renders a final scene constructed from ray-triangle intersections. For every pixel of the final image, a ray is shot against the scene made of triangles. The color of the pixel is the color of the triangle closest to the camera, or a predefined background color, if no triangle is intersected. Figure 2 shows a resulting render image from the ray tracing application of a cityscape model consisting of 11,826 triangles.

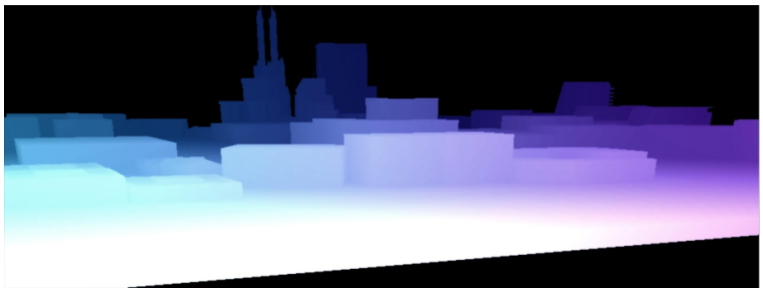

Figure 2: A rendering of a cityscape model

The application is organized into modules to facilitate its implementation by multiple developers simultaneously. Modular code also increases maintainability, making it easier to test and extend the project on successive semesters of development. There are four groups of modules used by the ray tracing application: *core components*, *camera*, *buffer*, and *raytracer*.

The *core components* group is composed of modules containing *abstract data types (ADTs)* essential to the other parts of the project. It includes the `VectorOverloading` module, responsible for implementing the `Vector2` (two-dimensional vector), `Vector3` (three-dimensional vector), and `Matrix3` (square matrix of third order) data types and operations involving them. Furthermore, it consists of the `Triangle`, `Ray`, and `IntersectionRecord` modules. Triangles are defined by three `Vector3` points *a, b, c*, representing their vertices, and a color attribute that stores its *RGB* information in a `Vector3`. Rays contain a `Vector3` point of origin, and another `Vector3` for storing its direction. Lastly, `IntersectionRecord` ADTs are used for recording intersection information. They store the distance between the intersection point and the ray origin (`Real`), the intersection point (`Vector3`), the normal of the intersected object (`Vector3`), and the color of such object (`Vector3`).

The *camera* group is composed of two modules: the orthonormal basis (`ONB`) and the `PinHoleCamera`. They are responsible for generating a ray for every pixel in the final image to be later intersected against

the list of triangles in the scene. `ONB` [9] is an auxiliary module whose task is to generate a matrix used for transforming a ray from the simplified camera space, where it is created, to the world space, where the scene (list of triangles) is located. In performing such a task, it uses `Vector3` and `Matrix3` operations defined in the `VectorOverloading` module. The `PinHoleCamera` module defines a simple camera without lens which uses perspective projection to depict rendered images. It contains attributes that define the characteristics of the final image (eg. `resolution_`), and the positioning of the camera (eg. `position_`, `up_`, `lookAt_`).

The *buffer* group consists of only one module of the same name. Its task is to manage file handling: reading from files containing triangle meshes of *obj* type to create a list of `Triangle` objects from it, and writing the resulting image from the ray tracing into a *ppm* file, which saves *RGB* values for every pixel. It stores the final image as a matrix of `Vector3` objects, each element representing the color of a pixel.

The *raytracer* group includes an assistance module, which utilizes the Möller–Trumbore intersection algorithm for handling ray-triangle intersections (`RayTriangleIntersector`), and the main module of our application, named `raytracer`. The Möller-Trumbore ray-triangle intersection algorithm is of particular importance as this is typically implemented in the imperative paradigm [5]. The main module uses all the other groups to generate an image product of ray tracing. It creates a camera with specifications for its positioning and the resolution of the final image. Then, it uses the camera to generate rays for every pixel, according to the previously defined parameters. For every pixel, it intersects the ray with the list of triangles extracted from the input *obj* file. Finally, it saves the result of all intersections into a *ppm* file, showing the renderization of the objects of the input. This particular process is typically implemented in C++.

Figure 3 summarizes the interaction between the groups of modules of the ray tracing application in the order of execution through a message sequence chart. The *core components* group is not referred to in the figure, since it consists of a core set of modules used by the other groups.

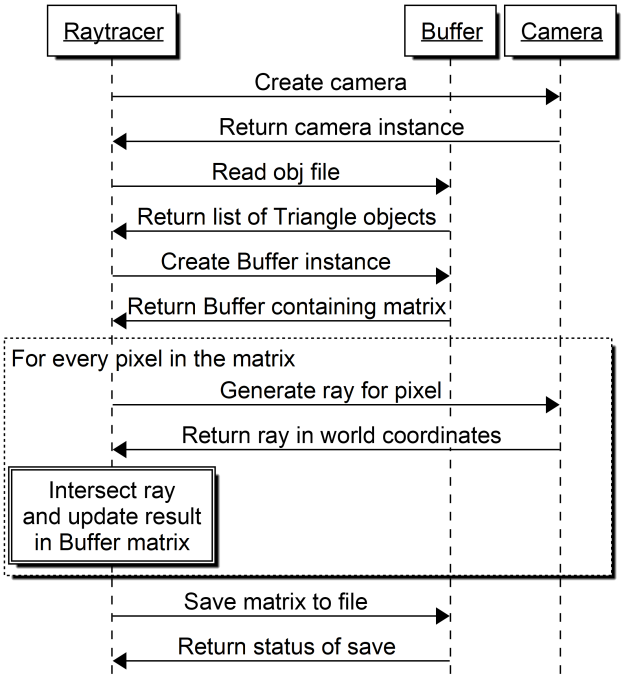

Figure 3: Ray tracing application message sequence chart

4 Integration into the Course

Having presented the ray tracing application, its integration into the functional programming course is presented in this section. Firstly, the strategy for integrating the application (4.1) is discussed. Then, the two stages (4.2) and two groups (4.3) used in the ray tracing application experiment are described. Special integrated exercises (4.4), with examples (4.5), are explained. Finally, ray tracing specific consultations (4.6) are shown, together with the feedback forms (4.7) used for improving the application on future iterations.

4.1 Comprehensive Method and Strategy

Starting from the beginning of the semester, we prepare the students for applying the concepts that they learn in an application. Homework problems are typically simplified versions of real life applications that are encountered in industry. Classwork problems are then respectively tailored to prepare the students for tackling the homework assignments. Additionally, during the lectures and practices it is emphasized when and where certain concepts are used in an application. In this manner, step by step and concept by concept we are able to show the students the reason for the material and its applicability.

As the course approaches the midterm and the concepts start becoming more abstract with the introduction of records and algebraic data types, we started transitioning the students towards the ray tracing project as the ray tracing application is a much more concrete method of demonstrating how these concepts come into play within the context of an application. Additionally, it is easier to demonstrate via an application how the flow of data and information is handled between functions and modules.

4.2 Stages of the Experiment

To assess the effectiveness of our application on the increase of interest for students, tests were performed in two stages during consecutive semesters. By doing so, not only did we increase the sample size of students, but also performed different ways of integrating the ray tracing project. Applying the experiment over two semesters allowed for testing with different students and comparing the outcome to determine the best integration approach.

For the first stage, we gave the students a brief lecture on the concepts of computer graphics and ray tracing before letting them work on the tasks. After reviewing the results from the first stage of the experiment, we started the second stage. It involved integrating the material into the course with an independent lecture on computer graphics and ray tracing, with lecture slides and study materials made available to the students. Such integration allowed them more time to better understand the task before having to implement them in their practical sessions and in their homework assignments.

4.3 Groups of Students

For the first stage of the experiment, as it took place during the spring semester, we had two groups of students. Since the majority of students take the course during the fall semester, the students who took the course during the spring semester were typically students who were retaking the course. These students generally had a harder time with the concepts of functional programming and overall less interest in the material. This formed our first group of students. To form our second group, we invited students who had completed the course during the fall semester who had demonstrated good performance during the course. This group generally had an interest in the material but had no plans to further pursue the subject proactively. After the completion of the first stage, we were able to perform another stage of the

experiment with the new batch of students in the fall semester. With an increased headcount and with the majority of the students being new students, we had no need to separate them into groups and treated the entirety of the class as a single group (third group).

4.4 Creation of the special exercises

The special exercises were created for the students to put in practice their current knowledge and to use their newly acquired skills in functional programming applications. For the exercises, we first implemented a finalized program based on Figure 3, which was able to render a red triangle as the default object. Then, the main testing idea was to give the students an incomplete version where the majority of the modules were already provided, only a small chunk of code was removed and then students were asked to implement the deleted parts. The deleted parts could be anything depending on which knowledge (e.g. List Comprehension, Pattern Matching, Tail Recursion, Records, and Abstract Data Type) we wanted to assess. We designed the tests so that, once our students fulfilled these parts, they would be able to produce a red triangle. Typically the students were provided a detailed description of the task, along with the function declaration, both of which are more than sufficient as hints to get the students started on the right path.

In terms of the content of the exercises, since they were the missing parts in the finished version, we created a collection of around ten potential questions, classified into three types:

1. Implementing underlying Abstract Data Structures (ADT) (e.g. `Vector2`, `Vector3`, `Matrix3`, and their operators). This class of questions showed our students the usage of records and ADTs.
2. Implementing the ray-triangles intersection related parts.
3. Handling input/output: reading points and polygon faces from obj files and writing into *ppm* files.

For composing a test, we first picked from the list above three questions which had an appropriate difficulty for that group based on their previous performance in classwork and homework assignments. The assignments varied according to the groups to better adapt to their experience in programming and, specifically, to the functional programming paradigm.

In the first and third groups of students, the first assignment involved instance realization (implementing multiple operators) for the complex type `Vector3`. The second task demanded the dot and cross product calculations for the same complex record type. Finally, `Vector3` normalization was requested as the third assignment. Such test involved easier questions from the exercises of type 1, since they were tailored to first-year first-semester students.

For the second group, the test comprised computing the determinant of a value of type `Matrix3`, implementing the ray-triangles intersection, and in the end, writing data to *ppm* files (questions of types 1, 2, and 3).

4.5 Example questions

For those students who found difficulty in grasping the concepts of functional programming, we only asked them to complete instances, functions for `Vector3`, `Matrix3` in the *core components* group, which only required a basic understanding of records and ADTs.

```
:: Vector3 a = {x0 :: a, x1 :: a, x2 :: a}

instance + (Vector3 a) | + a where + vector0 vector1 = {x0 = (vector0.x0 +
vector1.x0), x1 = (vector0.x1 + vector1.x1), x2 = (vector0.x2 + vector1.x2)}
```

```
instance - (Vector3 a) | - a where - vector0 vector1 = {x0 = (vector0.x0 -
vector1.x0), x1 = (vector0.x1 - vector1.x1), x2 = (vector0.x2 - vector1.x2)}
```

For average students, a wider set of topics could be inquired such as list comprehension, and tail recursion. For instance, with the loop in Figure 3, we demonstrated the usage of tail recursion through the intersection between the ray generated from one pixel and the list of triangles. Furthermore, we asked students to apply the intersection operation to obtain a complete buffer matrix in order to test their understanding of list comprehension.

```
// For one pixel
IntersectRayTriangles :: Ray [Triangle] IntersectionRecord →
    (Bool, IntersectionRecord)
IntersectRayTriangles ray_ triList initRec =
    IntersectRayTriangles2 ray_ triList (False, initRec)

IntersectRayTriangles2 :: Ray [Triangle] (Bool, IntersectionRecord) →
    (Bool, IntersectionRecord)
IntersectRayTriangles2 _ [] rec_ = rec_
IntersectRayTriangles2 ray_ [hTri:tTri] (bool_, irec_)
| newIRec.t_ > 0.0 && newIRec.t_ < irec_.t_ && bool_
= IntersectRayTriangles2 ray_ tTri (bool_, irec_)

// For all pixels
integrate resolution triangles = [[integrate2 j i triangles \\i←[1..x]] \\j←[y,y-1..1]]
```

Additionally, those students who could go further, challenged themselves with the `Buffer` module. It is especially demanding due to the complexity when using functional programming paradigm in file handling.

```
WriteFile :: *File [[Vector3 Real]] → *File
WriteFile file img
♯ file = file<<<"P3\n"
♯ file = file<<<length (img!!0)<<<' '<<<length img<<<'\n'
♯ file = file<<<255<<<'\n'
♯ file = foldl (λx y = x<<<y<<<' ') file img255
= file
```

4.6 Consultations

During the consultations that are offered during the course, we help the students with the material. Students are given help with implementing the code, although the assistance provided was limited to hints and suggestions, while implementing the missing pieces was left to the student. We also assisted the students with any mathematical challenges that they were facing in understanding the task. Additionally, we offered to teach more about ray tracing concepts to those who were interested in extra material.

4.7 Feedback forms

We conducted a questionnaire to collect relevant data about how the students evaluate introducing application oriented topics with its own specific exercises, homework assignments, classwork, assessment quizzes, and lab programming tasks. The goal was to test qualitative features like meaningfulness of the

topic for them. By ensuring anonymity of the respondents we maximized the comfort of the students answering.

5 Results

In this section, results comparing the performance of students from both semesters according to classwork (5.1), homework (5.2), and feedback forms (5.3) are presented and discussed.

5.1 Classwork

Since classwork is divided into a theoretical and practical part, two independent analyses were made to compare the integration of the ray tracing application between semesters. Relative metrics were considered to counter the difference in the number of students participating across the semesters. In the first stage, only 8 students participated against 55 in the most recent teaching period.

The performance for the theoretical part of the classwork (short quiz applied in the beginning of the classes) was consistent for both semesters. There was a one percent increase in the mean of performance of students, going from 63%, in the first stage, to 64% in the current one. However, two aspects were different in the two applications of the quizzes: the time limit and the phase of the course.

Students had ten minutes to finish a quiz in the first stage whereas pupils from the second stage had only eight (20% decrease). Furthermore, the test was applied at the end of the first stage, evaluating students that had already seen all the topics in the curriculum and were familiar with them. For students involved in the second stage of the experiment, the quiz of same size and difficulty was taken in the middle of the semester. Therefore, although current students had less time to finish the theoretical part of the classwork and it was applied at a much earlier stage in the course, they maintained the same performance as compared to their predecessors.

In the practical part of the classwork, the same questions were used for groups from each semester. Results displayed in Figure 4 show a significant improvement in the ability of students from the second stage in solving ray tracing related tasks, when compared to the ones from the first stage.

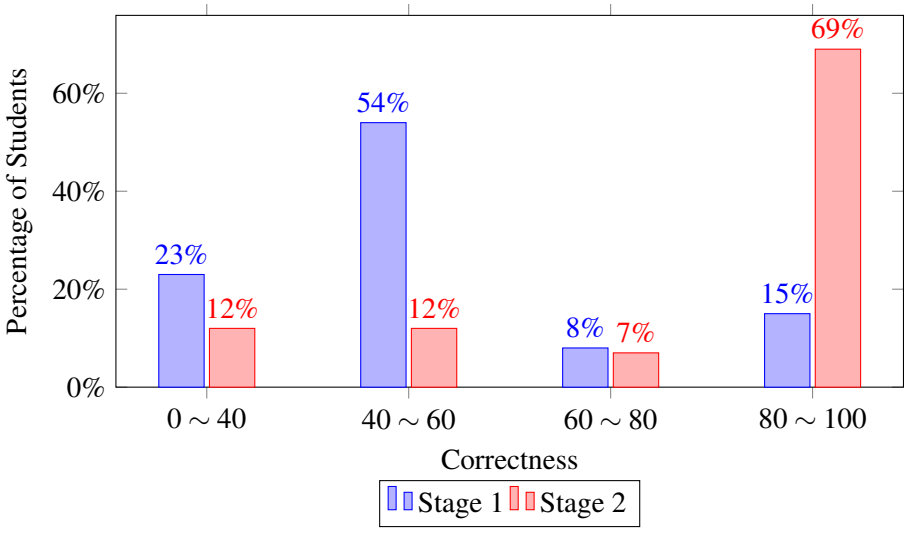

Figure 4: Classwork Comparison

Figure 4 shows that 69% of the students from the second stage achieved a score equivalent to the maximum grade (Excellent, above 80%) in the Hungarian education system, more than four times the numbers from the first stage. Moreover, the number of students failing (0-40 points) the practical part of the classwork decreased by roughly 50%.

It is important to state that, as in the theoretical counterpart, the practical part of the classwork was applied in the second stage at a much earlier point than in the first stage. In addition, the time limit for the practical part was the same in both semesters. These two conditions make the aforementioned increase in performance even more significant.

5.2 Homework

The analysis of homework scores is similar to the one done for the practical part of the classwork: students were given the same time frame to solve the problems of equivalent difficulty and length. However, no ray tracing related homework was applied in the first stage and, thus, no direct comparison could be made to the ray tracing homework done in the second stage. This happened since the limit of the numbers of homework assignments for the first stage had already been achieved when ray tracing concepts were presented at the end of the semester.

Be that as it may, a relevant comparison can be made to homework in which the same functional programming topics were included, and the problems count as well as the time limit for solving them are the same. Figure 5 shows a comparison between an equivalent homework from the first stage and the ray tracing related one administered in the most recent period of study.

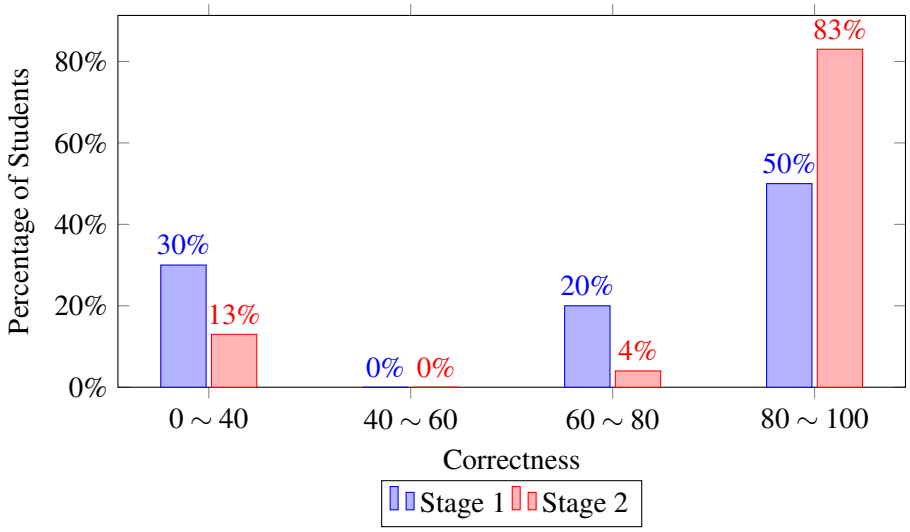

Figure 5: Homework Comparison

Figure 5 shows that the ray tracing integration into the homework improved the correctness of the assignments of students in a meaningful way. The greatest increase is seen in the range of 80 to 100 points, going from 50% to 83%. Additionally, there has been a decrease of more than half of the students who failed the assignment (0-40 points), even with more challenging problems for the homework applied in the second stage (Section 2.1).

5.3 Feedback forms

Feedback forms were collected and analysed identically for both stages of the experiment. The questions are listed below. Respondents could reply with either "Disagree", "Agree", or "Neutral". Figures 6 and 7 show the responses of the students from stage 1 and stage 2 respectively.

- Q1: Ray tracing application via functional programming is relevant to my interests.
- Q2: Ray tracing implementation in function programming is sufficiently difficult for me.
- Q3: Applying functional programming as part of a larger project helps with understanding functional programming concepts.
- Q4: I enjoyed implementing the code and was motivated to do so.
- Q5: Ray tracing has helped my curiosity about functional programming.
- Q6: Integration of ray tracing into the curriculum helps demonstrate functional programming possibilities.
- Q7: Enough background was provided to allow me to understand the exercises.
- Q8: The exercises assigned to me were challenging.
- Q9: I would be interested in taking an application based advanced Functional Programming course.

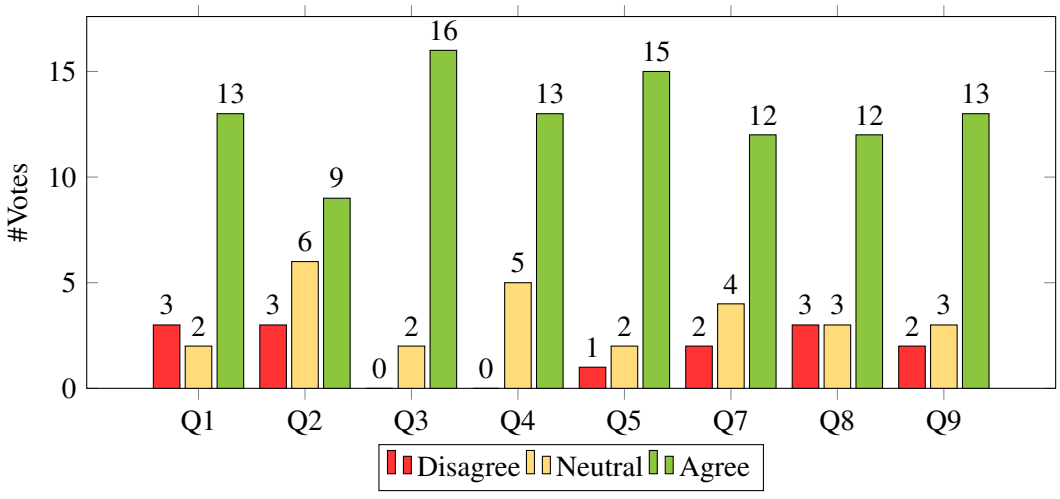

Figure 6: Stage 1

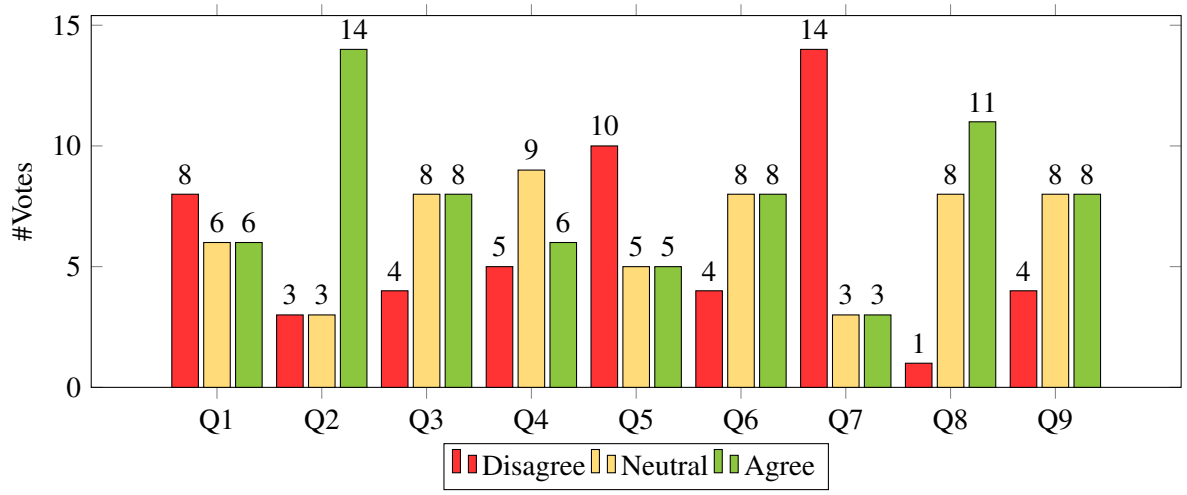

Figure 7: Stage 2

The histograms in Figures 6 and 7 show us several points:

1. The majority of students find it sufficiently difficult implementing ray tracing in functional programming.
2. In the second stage, most of the students believe that they don't get enough background to understand the exercises.
3. There are more students who are interested in taking an application based advanced Functional Programming course than students who don't.
4. Most students approve the application of functional programming as part of a larger project. They defend the position that it helps with understanding functional programming concepts.

We received in-person feedback from eight of the students in the first stage of the experiment, demonstrating interest in further pursuing learning and developing applications in functional programming. This is double the amount of students who expressed the same interest in the previous year, before the experiment. As of the time of writing, the students are now in progress of doing a research project using Clean to develop a sound synthesizer application (6.3).

The feedback questionnaire presented results indicating that students find functional programming to be better taught when applied to larger projects. It also showed that they enjoyed implementing the ray tracing related exercises, and it has piqued their curiosity towards the functional paradigm, inspiring ideas for new applications (Figure 6).

It is noticeable by the feedback data that students who perform excellently on the subject of functional programming are even more interested by its applications when compared to regular alumni. This shows that applying functional programming to ray tracing not only piques the interest of regular students, but also provides new challenges to keep excellent students motivated (Figure 6).

However, compared to the first test groups, the second test groups expressed frustration at the difficulty of the tasks and felt that they did not have enough background or proficiency to tackle the tasks (Figure 7). This is likely due to the difference in exercises provided to the students between the first stage and the second stage.

A possible reason for the difference in the two stages perhaps is due to our integration. In the first stage, the experiment was conducted near the end of the semester. As such, the students already had

more experience in the course and the tasks given to them were less demanding. With the integration into the course from the second stage, the students were given more demanding tasks at an earlier point in the course. This was most likely the cause of their feelings of being overwhelmed by the material and tasks.

All in all, every group showed improvement in their understanding of functional programming and a marked increase in their interest in further pursuing related projects and advanced courseworks.

6 Future Work

With the benefit of hindsight on our experiment, we found multiple points of possible improvement and extension for the future. In this section, ray tracing application improvements (6.1), the integration of any application into the functional programming course (6.2), and the development of other applications (6.3) are discussed.

6.1 Ray Tracing Application Improvements

For the ray tracing application, there are significant areas for improvement and expansion. One immediate area for improvement would be to refactor the code for easier readability, better comprehension, and increased efficiency. Additionally, we can add more features to extend the application. One notable feature would be the use of texture files. Another further addition would be the use of multiple lighting sources or lighting types.

6.2 Application Integration

For better integration of the application into the curriculum, we think it would be beneficial to do some improvements. One would be the creation of a web based interactive compiler that would allow the students to solve the exercises, compile, and instantly see their results in a controlled environment akin to the HackerRank model [4].

6.3 Other Applications

In the future, other applications involving functional programming should also be explored. Currently the team is working on a new application involving sound synthesis, and hopefully with more interest from the students a wider variety of applications can be explored. This in turn can cultivate interests in more students and motivate them to further pursue functional programming.

7 Conclusion

Through our experience in teaching functional programming within the given curriculum constraints, we came up against multiple challenges in cultivating an interest in our students and motivating them to further pursue functional programming. We decided upon utilizing an application programmed entirely in the functional paradigm, which inspired students in many ways. They were excited and more self-motivated for learning functional programming and had increased engagement when compared to earlier practical sessions. Since it is widely used in multiple fields and uses multiple functional programming concepts, ray tracing was an attractive application overall for the students. The questions related to ray tracing suited the entirety of the semester, thus being a valuable addition to the curriculum.

Acknowledgement

This work was supported by the European Union, co-financed by the European Social Fund, grant. no **EFOP-3.6.3-VEKOP-16-2017-00002**. We would also like to extend our eternal gratitude to the students taking part in the teaching experiments and to the demonstrators helping in the preparation and carrying out of the survey.

References

[1] Clean Language Report, `https://clean.cs.ru.nl/download/doc/CleanLangRep.2.2.pdf`, Accessed 3 April 2019.

[2] Gerdes A., Heeren B., Jeuring J.: Teachers and Students in Charge, In: Ravenscroft, A. et al.: *7th European Conference on Technology Enhanced Learning*, LNCS, Vol. 7563, Springer Verlag, 2012, pp. 383-388, doi: 10.1007/978-3-642-33263-0_31.

[3] Glassner, A.S.: An Introduction to Ray Tracing, *Academic Press Ltd.*, London, UK, 1989.

[4] HackerRank homepage, `https://www.hackerrank.com/`, Accessed 10 October 2019.

[5] Möller T., Trumbore B.: Fast, Minimum Storage Ray-Triangle Intersection, *Journal of Graphics Tools 2.*, pp. 21–28, doi: 10.1080/10867651.1997.10487468.

[6] Saidani, T.: Optimisation multi-niveau d'une application de traitement d'images sur machines parallèles, Ph.D. Thesis, Université Paris-Sud XI, 2012.

[7] Thompson, S.: Haskell: The Craft of Functional Programming,*Addison-Wesley Longman Publishing Co.*, Inc. Boston, MA, USA, 1999.

[8] Trinder, P.W., Hammond, K., Loidl, H-W., Peyton Jones, S.J.: Algorithm + Strategy = Parallelism, *Journal of Functional Programming*, Vol. 8, No. 1, Cambridge University Press, January 1998, pp. 23–60, doi: 10.1017/s0956796897002967.

[9] Pedro Henrique Villar de Figueirêdo: Relato de Experiência Sobre o Aprendizado de Introdução à Renderização Baseada em Física em um Curso de Graduação da Área de Computação, *Comunicações em Informática*, Vol 1, Nr. 1, 2017, pp 18-21, doi: 10.22478/ufpb.2595-0622.2017v1n1.37497.

[10] Zsók V., Hernyák Z., and Horváth, Z.: Designing Distributed Computational Skeletons in D-Clean and D-Box. In: Horváth Z. (ed.): *Central European Functional Programming School*, CEFP 2005, First Summer School, Budapest, Hungary, July 4-15, 2005, Revised Selected Lectures, LNCS Vol. 4164, Springer-Verlag, 2006, pp. 223–256, doi: 10.1007/11894100_8.

[11] Zsók V., Horváth Z.: Intensive Programmes in Functional Programming, In: Morazán, M.T. et al.: *The International Workshop on Trends in Functional Programming in Education*, TFPIE 2012, St. Andrews, Scotland, June 11, 2012.

Appendix A Implementation

A.1 VectorOverloading

```
:: Vector3 a = {x0 :: a, x1 :: a, x2 :: a}

instance == (Vector3 a) | == a
instance zero (Vector3 a) | zero a
instance one (Vector3 a) | one a
instance ~ (Vector3 a) | ~ a
instance + (Vector3 a) | + a
instance - (Vector3 a) | - a
instance * (Vector3 a) | * a
instance / (Vector3 a) | / a

Vec3dotProduct :: (Vector3 a) (Vector3 a) → a | *,+ a
Vec3crossProduct :: (Vector3 a) (Vector3 a) → (Vector3 a) | *,-a
Vec3normalize :: (Vector3 a) → (Vector3 a) | sqrt,*,+,/ a

:: Vector2 a = {v0 :: a, v1 :: a}

instance == (Vector2 a) | == a
instance zero (Vector2 a) | zero a
instance one (Vector2 a) | one a
instance ~ (Vector2 a) | ~ a
instance + (Vector2 a) | + a
instance - (Vector2 a) | - a
instance * (Vector2 a) | * a
instance / (Vector2 a) | / a

:: Matrix3 a = {a0 :: a, a1 :: a, a2 :: a, b0 :: a, b1 :: a, b2 :: a, c0 :: a, c1 :: a, c2 :: a}

Mat3determinant :: (Matrix3 a) → a | *,-,+ a
Mat3Vec3Product :: (Matrix3 a) (Vector3 a) → (Vector3 a) | *,+a
```

A.2 ONB

```
:: ONB = {u_ :: (Vector3 Real), v_ :: (Vector3 Real), w_ :: (Vector3 Real), m_ :: (Matrix3 Real)}
setFromUW:: (Vector3 Real) (Vector3 Real) → ONB
setFromV::(Vector3 Real) → ONB
```

A.3 Ray

```
:: Ray = {origin_ :: (Vector3 Real), direction_ :: (Vector3 Real)}
```

A.4 IntersectionRecord

```
:: IntersectionRecord = {t_ :: Real, position_ :: (Vector3 Real),
normal_ :: (Vector3 Real), color_ :: (Vector3 Real)}
```

A.5 Triangle

```
:: Triangle = {colorT_ :: (Vector3 Real), a_ :: (Vector3 Real),
b_ :: (Vector3 Real), c_ :: (Vector3 Real)}
(intersect) :: Ray Triangle → (Bool, IntersectionRecord)
```

A.6 RayTriangleIntersector

```
IntersectRayTriangles :: Ray [Triangle] IntersectionRecord → (Bool, IntersectionRecord)
```

A.7 Buffer

```
InitBuff :: Int Int → [[Vector3 Real]]
GetBuffElem :: [[Vector3 Real]] Int Int → Vector3 Real
SetBuffElem :: [[Vector3 Real]] Int Int (Vector3 Real) → [[Vector3 Real]]
SaveToPPM :: [[Vector3 Real]] String *World → *World
LoadObj :: String *World → ([Triangle], *World)
```

A.8 PinHoleCamera

```
:: PinHoleCamera = { minX_ :: Real,
    maxX_ :: Real,
    minY_ :: Real,
    maxY_ :: Real,
    distance_ :: Real,
    resolution_ :: Vector2 Int,
    position_ :: Vector3 Real,
    up_ :: Vector3 Real,
    lookAt_ :: Vector3 Real,
    direction_ :: Vector3 Real,
    onb_ :: ONB }

initCamera :: Real Real Real Real Real (Vector2 Int) (Vector3 Real) (Vector3 Real) (Vector3 Real)
→ PinHoleCamera
getWorldSpaceRay :: PinHoleCamera (Vector2 Int) → Ray
```

A.9 RayTracer

```
MAX_REAL =: 999999999999.9
resolution :: Vector2 Int
camera_position :: Vector3 Real
camera_up :: Vector3 Real
camera_lookAt :: Vector3 Real
camera :: PinHoleCamera
integrate :: (Vector2 Int) [Triangle] → [[(Vector3 Real)]]
integrate2 :: Int Int [Triangle] → Vector3 Real
```