Ray Tracing Motivating Application in Teaching Functional Programming

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

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

pedrofigueiredo5206@gmail.com, kimyuri31@gmail.com, aaogmu@inf.elte.hu

Sitt.Evan@gmail.com, xueying19981206@gmail.com, zsv@inf.elte.hu

Ray tracing is an intensively studied field of computer graphics and it is widely applied in current day media. Functional programming represents a modern tool for applying and implementing software for information and society. 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 fill the gap between theoretical functional programming teaching and its applications. The educational question addressed is whether computer graphics, more specifically ray tracing, increases the level of motivation of students for studying the functional programming paradigm. We conducted this teaching experience and we had positive results and feedback from our students, described here in the paper.

1 Introduction

This paper reports a teaching experience of using practical solutions to make the functional programming course more attractive for first year, first semester students. In recent years, functional programming has become among the first major programming language courses [2, 7, 10]. However, 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 steps. This causes the student to become unable to progress in later classes, since the curriculum is fast-paced and iterative.

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 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.

We considered presenting an especially attractive computer graphics application - ray tracing - wherein, after short introductory presentations, they can extend the Abstract Data Type (ADT) with new functions and try out a well-known concept, very used on popular entertainment products such as movies and games. Ray tracing usually is studied extensively in the framework of more advanced courses. However, here we provided the implementation for the complex parts, leaving it up to the students to extend it by the minor missing properties or operations required to visualize graphical elements. This way, they were able to practice various concepts in the scope of the same application.

In this paper, we describe the Clean implementation of the Ray Tracing solution (section 2), explain the functional programming course's structure (section 3), report the exercises and learning situations we placed the students in to increase the functional programming learning experience (section 4) and analyze the results (section 5), and finally we conclude with future work (section 6).

Submitted to:
TFPIE 2019

1.1 Challenges of Teaching Functional Programming

At Eötvös Loránd University, the course taught during the Bachelor's studies is an introductory, compulsory, approach into the functional programming paradigm. We utilize the functional language Clean (also known as Concurrent Clean [1]) to demonstrate these concepts to the students.

Concurrent Clean is a uniquely typed purely functional programming language. Being a purely functional language, it is an effective tool for teaching the paradigm of functional programming as it is strongly-typed, generating more compilation errors rather than execution ones.

Even though having a strongly-typed language has many advantages, which are most appreciated by those who have experience in such programming style, it can be unclear for first-year students. This happens usually due to the reduced size of projects taken by beginners and the learning curve to implement simple examples from other object-oriented or procedural languages in the functional paradigm.

A simple example of something difficult to convey to students is why a simple piece of code within C++ or JAVA such as `i=i+1;` is not possible within a functional language. Simple statements such as this are so well ingrained within the students' minds that it usually takes quite some time for them to embrace immutable data structures.

Another common issue is showing the difference in syntax and structure of functional code. For example, a common use case scenario in C++ for a function creating a list of numbers, as seen on the Figure 1.

```
std::vector<int> foo(int x){
  std::vector<int> bar;
  for(int i=1; i≤x; i++)
  {
    bar.push_back(i*i);
  }
  return bar;
}
```

```
foo:: Int → [Int]
foo x = [i^2\\i←[1..x]]
```

Figure 1: C++ vs Clean

Having stated that, it can be challenging to teach functional programming to first year, first semester students. Therefore, finding an application, such as ray tracing, to demonstrate a use-case of the language could help motivate students to overcome the mentioned challenges.

1.2 Ray Tracing Motivation

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 2). It is considered a heavy computational algorithm with approximate results since the number of light interactions in an environment is infinite. This is the reason why lower level languages are typically usually used for solving this problem [4].

Its inherently interdisciplinary background, covering multiple topics from Mathematics and Physics, has shown to be a motivational application of theoretical concepts which are considered hard to understand according to undergraduate students. Moreover, its realization encompasses numerous sub-

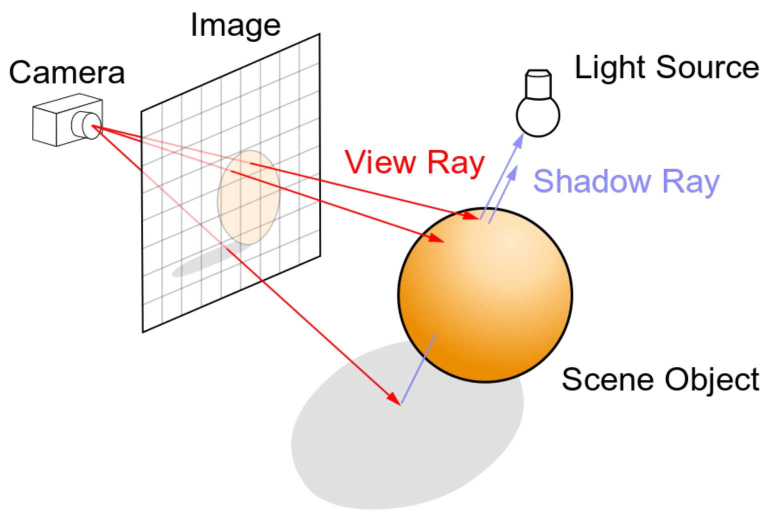

Figure 2: Ray tracing diagram

problems, varying in size and complexity. This natural subdivision facilitates its implementation on different levels throughout the semester of a Bachelor's level course. Beginners could help tackle simple algebraic operations while more experienced students would contribute with 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 palpable objective: generating images as a result of the ray tracing rendering.

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, 9]. With Ray tracing, considered to be a "embarrassingly parallel" [4] problem, the implementation of such features as well as the understanding of students would be facilitated.

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.

Finally, as it is currently used within a multitude of popular products such as movies, television series, and games, ray tracing [4] can have a motivational appeal towards students. Realizing that, by learning functional programming concepts, one can be a part of a ray tracer software project brings meaningful examples of their importance. It is especially effective as it uses the same basic concepts of other state-of-the-art physically-based renderers, namely the Arnold Renderer, developed by Solid Angle and utilized on movies such as Logan (Figure 3) and Thor: Ragnarok, and on world famous television series Game of Thrones [4].

2 Implementation of Ray Tracing in Clean

This section provides brief explanations on the modules used by the ray tracer application in the Clean programming language. The full implementation of the modules described in the following subsections are available on annex A.

Figure 3: Comparison between reference actor (left) and rendering result (right) on movie Logan.

2.1 Vectors in Clean

`Vectoroverloading` is a base module that is used for this project. It contains the types for 2-dimensional vectors, 3-dimensional vectors, and 3-by-3 matrices, named `Vector2`, `Vector3` and `Matrix3x3` respectively. Some instances for basic operators such as ==, `zero`, `one`, `~`, `+`, `-`, `*`, `/` are defined for `Vector2` and `Vector3`. There are further functions defined for `Vector3` such as the `Cross product`, `Dot product`, and `Normalizing`. For `Matrix3x3`, we have a function which calculates the determinant. An additional product function that works between `Matrix3x3` and `Vector3` is also defined. As this module corresponds to basic algebraic operations, all other modules depend on it.

2.2 Buffer

Buffer is the module responsible for file handling. Since Clean models outside communication through `*World`, which is an `ADT`, it makes communication more efficient. We used the let-before expression as suggested by the Clean Language Report [1]. When reading an obj file, we extract useful information from the input such as vertices and faces (list of vertex indices to form a plane). After extracting vertices and faces, it is possible to add color and generate triangles (and, consequently, any other shape) from this data.

Another goal of this module is to write the result of a rendering to a ppm file (for convenience, since ppm files are composed of rgb-like structures). There are many approaches to printing out the result (i.e multiple let-before expressions, recursion, and others) but they are not from the functional programming paradigm. In search for readability, an imperative style is inevitable for this case. We chose a solution that used extensively optimized and validated built-in functions from Clean and still kept the code readable, as can be seen on the function `WriteFile` below.

```
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
where
```

```
img255 = (flatten o flatten) (map (λrow = map (λelem = to255 elem) row) img)
to255 {v} = [cvt v.x0, cvt v.x1, cvt v.x2]
cvt x
| x < 0.0 = 0
| x > 1.0 = 255
= toInt (x * 255.0)
```

2.3 Fundamental Components

A good example of the use of ADTs is implementing Ray, Triangle and Intersetion Record types for this project. They are fundamental types used throughout the other modules. `Ray` defines the essential representation of a ray of light which is the basic component of ray tracing. `Triangle` defines the triangle primitive by saving its corners in `Vec3`. Finally, the `IntersectionRecord` type carries information used in the ray-triangle intersection procedure.

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

:: Triangle = {colorT_ :: (Vector3 Real), a_ :: (Vector3 Real),
b_ :: (Vector3 Real), c_ :: (Vector3 Real)}

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

2.4 ONB and Pin Hole Camera

ONB is used to build a coordinate system for the camera. Our ONB can be set up in two ways. In the first method, we set up an ONB from the normalized input vectors `u` and `w`, which corresponds to `u` and `w` vectors of the ONB (`setFromUW:: (Vector3 Real) (Vector3 Real) → ONB`). In the second method, we set up an ONB from a normalized input vector `v` aligned to the `v` (up) vector (`setFromV::(Vector3 Real) → ONB`).

Afterwards, our system in Clean is built (orthonormal basis: u_, v_, w_):

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

The Pin Hole Camera module defines and instantiates the camera from which rays will be shot against the scene to produce the final image. We chose to use the Pin Hole implementation instead of a simpler orthographic solution to be able to display perspective on the final result. It uses the ONB record and is denoted by:

```
:: 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 }
```

Its most important goal is to generate a world space ray, given a screen pixel coordinate, which is implemented in `getWorldSpaceRay :: PinHoleCamera (Vector2 Int) → Ray`.

2.5 Ray Triangle Intersection

Ray triangle intersection is an essential part of the ray tracing application, as rays need to be intersected with the triangles determined by the mesh input files and stored in the buffer. We accomplish this task via the Möller–Trumbore ray-triangle (M-T RT) intersection algorithm [5]. Typical implementations of the M-T RT intersection algorithm is typically done within C++ making use of multiple if-else clauses. This is reproduced in Clean via pattern matching.

2.6 Ray Tracer

The Ray Tracer module connects all of the previously shown functions and modules to produce the final image using `integrate :: Resolution [Triangle] → [[Color]]` where Resolution is of type `Vector2 Int` and Color is of type `Vector 3`. The function takes the resolution of the generated image and the list of faces of a 3D model as input parameters. It then applies our ray tracing algorithm to obtain the color of each pixel for the result image. Examples of resulting images produced by this module can be seen on Figures 4, 5, and 6

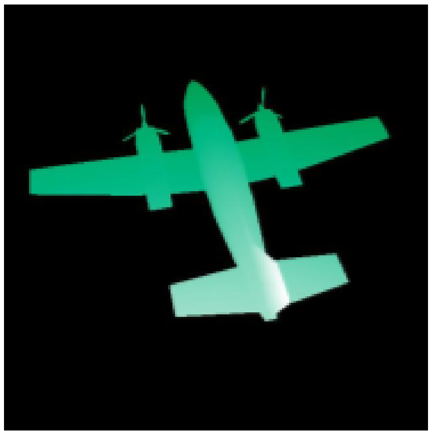

Figure 4: Airboat

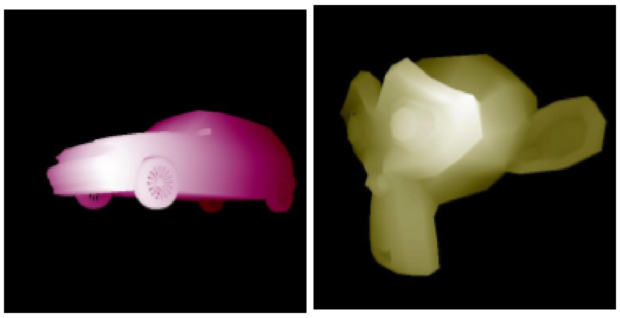

Figure 5: Fiat and Suzanne

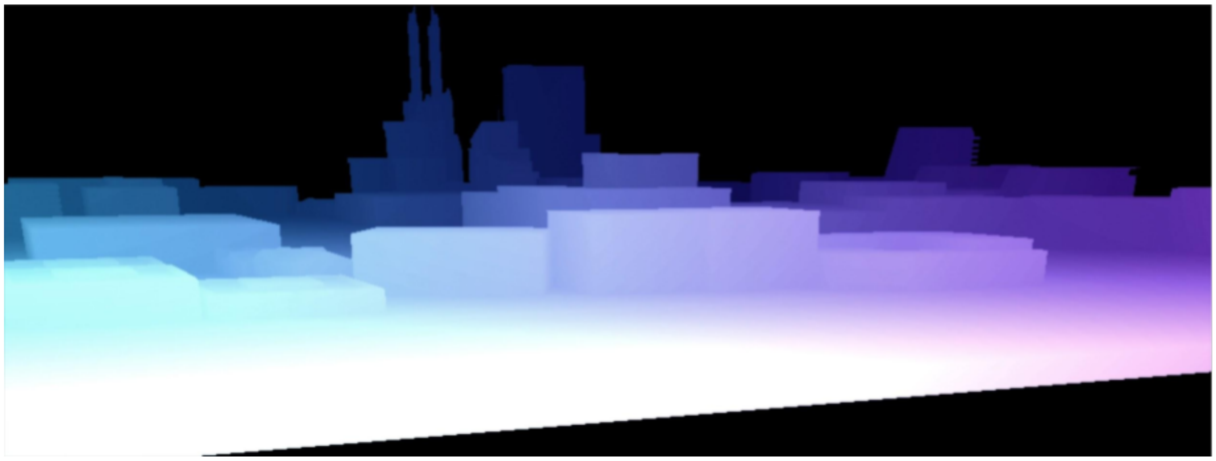

Figure 6: City

3 Ray tracing applied in functional programming teaching

This section presents the format of the introductory Functional Programming course at Eötvös Loránd University. We then present how our ray tracing application can help demonstrate the functional programming concepts and skills presented throughout the duration of the course.

3.1 Functional Programming Class Structure

The Functional Programming course is divided into two components. There is a lecture component and a practical component. While the lecture component is composed by presentations focused on the theoretical part, the practical component uses code snippets as its main driver.

There are twelve lecture sessions for the course, one per week during the duration 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.

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. 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.

In addition to the compulsory lecture and practical sessions each week, a consultation session is offered at the end of each 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 learn new methods and techniques in functional programming, along with emphasizing good programming form.

Furthermore, one of our teaching assistants also conducts a more specialized type of consultation via online streaming. Since the students have a self organized Facebook study group, it became an effective method to appeal to modern technology and trends and conduct consultation online. This had 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 reading the code being presented by the teaching assistant as it was on their own screen. Questions were also posed easier as any questions a student would have could be sent to chat without disrupting the teaching assistant or drawing attention, and could be addressed by the teaching assistant as appropriate. Lastly, with the nature of a digital format, the codes 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 have less 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 are 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, along with prepare 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.

3.2 Applying Ray tracing

Using the application of ray tracing, we were able to demonstrate to the students the application of many functional programming fundamentals in action. Nevertheless, one of the most important things for freshmen is to inspire them about the field they are studying. Thus, we let them explore what they can build from the knowledge they obtained. Typically, students have many questions about the subject regarding its impact in their future. Hence, we present this project to not only summarize the knowledge of the semester but also motivate them in their further learning path.

Records, classes, and abstract data types were well demonstrated in our exercises for the students involving implementation of the `Vector3`, `Vector2`, and `Matrix3` classes and their associated instances. As these three classes were the foundation for the entirety of the application, it helped present the crucial importance of implementing them and their abstraction.

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

instance ==   (Vector3 a) | == a   where == vector0 vector1 = vector0.x0 == vector1.x0
&& vector0.x1 == vector1.x1 && vector0.x2 == vector1.x2

instance zero (Vector3 a) | zero a where zero = {x0 = zero, x1 = zero, x2 = zero}

instance one  (Vector3 a) | one a  where one  = {x0 = one, x1 = one, x2 = one}

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

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)}
...
```

Pattern matching could be demonstrated as being an effective technique for reading data from an .obj file. As shown in the preprocess function, pattern matching is excellent for parsing strings read from a file. Every line starts with a character which defines whether it is a face, a vertex or something else and we have to classify and process each of them separately.

```
preprocess :: [(Char, String)] → [(Char, [[Char]])]
preprocess [] = []
preprocess [(t, 'v') : xs] = [(t, resv) : preprocess xs] // vertices
preprocess [(t, 'f') : xs] = [(t, resf) : preprocess xs] // faces
= preprocess xs
...
```

List comprehension showed its true power in a magnificent manner when it could be shown how a typically long doubly nested for-loop in standard C++ implementation could be simplified into a single line nested list comprehension.

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

Tail optimized recursion and tuples played a huge part in the ray tracer application in determining the intersection between a ray and the closest model polygon face. This was a suitable example to see the difference in computation time between recursion and tail recursion due to the enormous amount of polygons.

```
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_ && newBool == True
= IntersectRayTriangles2 ray_ tTri (newBool, newIRec)
= IntersectRayTriangles2 ray_ tTri (bool_, irec_)
```

```
...
```

Last but not least, we guided students to develop an instinct about the appropriate situations to use Functional programming style, not just blindly applying it. In the I/O section of the code, due to the complexity when using only FP approach, students will learn how to flexibly combine many programming styles.

```
LoadObj :: String *World → ([Triangle], *World)
LoadObj fname w
♯ (ok, file, w) = fopen fname mode w
| not ok = abort "Cant open"
♯ (content, file) = ReadFile file
...
♯ (content, file) = ((CreateTriangle o splitVF) content, file)
♯ (ok, w) = fclose file w
| not ok = abort "Cant close"
= (content, w)
```

Each of these concepts have been presented to the students over the course of the semester in various lectures and practical sessions as individual fundamentals. The ray tracer application was an effective method of presenting to the students how putting together even just the basic fundamentals they learned in class with the skills emphasized within the homework exercises, they could build a complex application to render computer based graphics. In addition, ray tracing is an intuitive topic for introducing parallel concepts.

4 Evaluation

This section presents the evaluation plan for students. It is separated in two iterations, each with a different set of assignments for groups of students with different backgrounds and levels in functional programming.

4.1 First Iteration

In a common practice class, towards the end of the semester, students were introduced to the idea of ray tracing with a short 30 minute lecture encompassing main points for overall understanding. The lecture material covered basic topics from ray casting and computer graphics, such as ray-primitive intersection, and geometric transformations.

Immediately after, they were part of a 30 minute test composed by three assignments organized in ascending level of complexity. The first one 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.

We evaluated the assignments based on their complexity, using different weights to each level of difficulty, and summing up to a final score. Upon gathering the results of the evaluation, we decided to have a new iteration, with more complex assignments and test in a set of more experienced students.

4.2 Second Iteration

The second iteration has a new set of assignments to increase the amount of students to be analyzed. This new trial targets current first year, first semester students as well as others that have passed with maximum grade, composing two different sets of participants.

The test is composed by a feedback quiz and a set of three new assignments. The feedback quiz concerns the feedback of students on the ray tracing application approach as a motivational tool. It aims at confirming our assumption based on observation that students are more motivated with tasks applied to ray tracing when compared to regular ones. The assignments vary according to the group to better adapt to their experience in programming and, specifically, to the functional programming paradigm.

The feedback questionnaire was composed by affirmations. Students would rate their level of agreement in regards to each affirmation in a scale from 1 to 5, with "Strongly Disagree" being represented by 1, and "Strongly Agree", represented by 5. This scale allows easy analysis of the data, since it is a numerical scale, being easily reproduced in charts. When compared to the binary (Agree or Disagree) alternative, it provides the student with more options and avoids bold assumptions based on imprecise data. Upon evaluation of assignments of students, we gathered and compared the data. In order to analyze their performance and motivation, we processed their feedback into charts for better visualization, which are presented and discussed in Section 5.

5 Results

We expected that by implementing the Ray Tracer application in Clean, the process would pique the interest of the students for learning more about functional programming and functional languages. After applying the assignments and quizzes to the students, we got their feedback and are able to establish relations between our expectations and what did they accomplish or how did they feel after such activities.

5.1 First Iteration

The first iteration consisted of ray tracing-based practical exercises and was applied to first-year, first-semester students.

5.1.1 Ray Tracing Applied Assignments

The assignments produced a mean score above the regular score obtained on same level exercises in previous occasions by the same students.

Since it is the first time for most of them to learn this concept, we could see their confusion. But they tried so hard to do the assignments and even asked for help in the consultation. According to the results, there were more than half of students (57%) done the around 50% of assignments.

However, the final result was not satisfactory, achieving a mean of only 42%. Also, it did not contemplate students from different levels and their feedback was not accounted. Thus, a new iteration was formulated, with new tasks, feedback analysis and a wider range of students to be evaluated.

5.2 Second Iteration

The second iteration consisted of ray tracing-based practical exercises and an anonymous feedback questionnaire. It was applied to first-year, first-semester students (group 1) as well as students from previous semesters (group 2) that have achieved an excellent mark on both lecture and practice courses of functional programming.

5.2.1 Ray Tracing Applied Assignments

For group 1, we compared the data with their first iteration. The good news is the mean value increased by 17% and 29% of students completed more than 75% of the assignments. And surprisingly, there is one student who got all tasks correctly done, while in the first iteration the highest correctness is only 53%.

It is clear that there is a significant change in the number of students who can handle most of the assignments, which is not too unexpected since we could feel their passion in this project from the students who came to the consultation for this projects.

For group 2, since they have a good fundamental of functional programming, we gave some more creative and challenging tasks for them. Even though the process is difficult and we could see they were struggling in these 2 hours, the final results are unexpected good. All of 12 students done 100% assignments and they were so excited when they saw a beautiful triangle constructed on the screen in the end.

5.2.2 Feedback Questionnaire

The feedback questionnaire presented results indicating that students find functional programming to be better taught when applied to larger projects (Figure 7). It also showed that they enjoyed implementing the ray tracing applied exercises (Figure 8), and it has piqued their curiosity towards the functional paradigm, inspiring ideas for new applications (Figure 9). Results are separated in two groups: group 1 referring to the first-year, first-semester students; and group 2 composing students of previous semesters which got an excellent mark. Charts are composed by groups of numbers representing the level of agreement with the statement chosen by participants, as explained in Section 4.2. Levels not chosen by any student are not displayed in the charts.

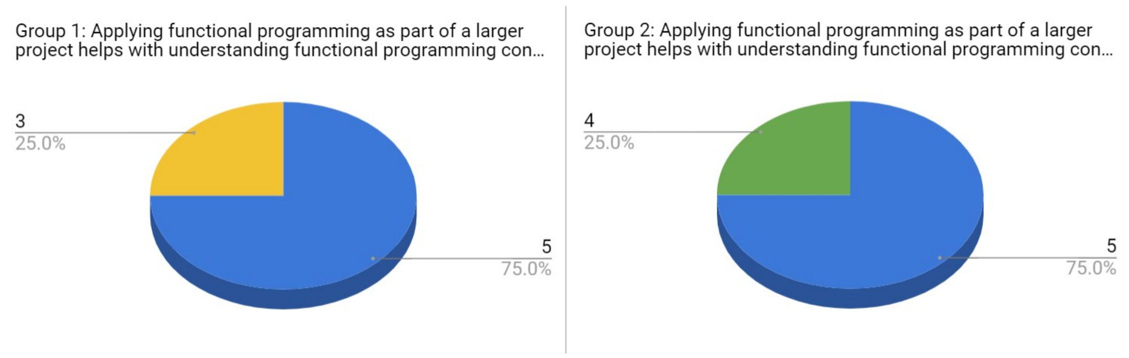

Figure 7: Results for statement: "Applying functional programming as part of a larger project helps with understanding functional programming concepts."

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.

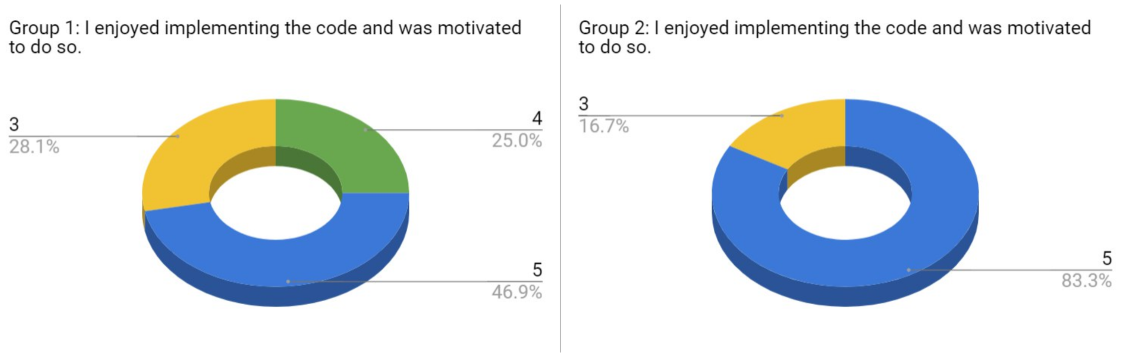

Figure 8: Results for statement: "I enjoyed implementing the code and was motivated to do so."

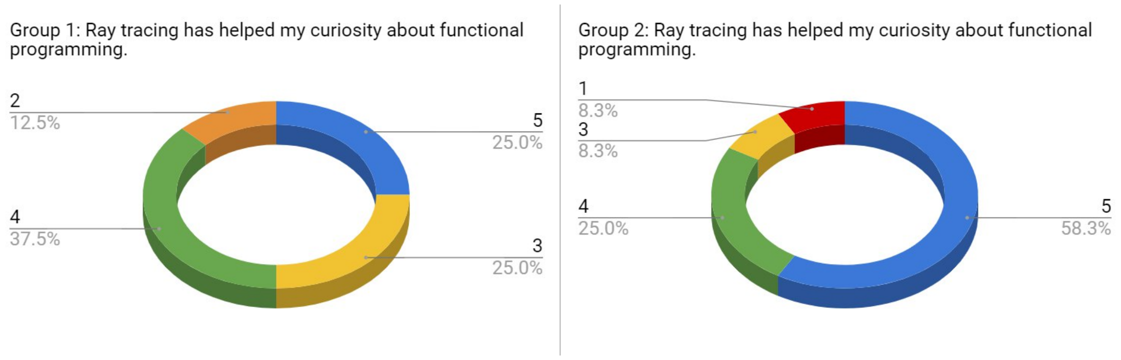

Figure 9: Results for statement: "Ray tracing has helped my curiosity about functional programming."

5.3 Perspective of the lecturer

Even though most students did not complete all the assignments due to the time limit, it was noticeable that they did not hesitate on trying until the end of the class, which is not the usual attitude other encounters. Furthermore, there was a significant increase in the number of students during consultation hours on the same week. Most students were interested on solving the ray tracing tasks rather than non-related ones.

Responses from both group one and group two agreed that the tasks were challenging for them to implement and were helpful for understanding the concepts of functional programming. Both groups one and two also responded that they were motivated to implement the tasks provided to them. Interestingly enough, the low performing group two replied much more positively than group one in terms of their interest in what they were doing. This was a stark contrast to their general indifference to the topics and small tasks of the course.

6 Conclusion and future work

The functional programming teaching experience inspired students in many ways. They were excited and more self-motivated for learning functional programming and tried to participate more than at earlier practical sessions. Since it is widely used in the entertainment media and uses multiple functional programming concepts, ray tracing was an attractive application overall for the students, as it could be seen in section 5. The questions related to ray tracing suited the semester ending topics of abstract data type and more complex data structures, thus being a valuable addition to the curriculum.

New iterations on next semesters of functional programming classes could prove very important to further validation and improvement on the use of applications such as ray tracing. Starting a semester already applying initial concepts to complete ray tracing projects, putting their abstract knowledge to real-world use, could avoid the high rate of drop out on the first weeks of the course. Also, applying feedback questionnaires at different stages of a semester would help evaluate the effectiveness of such application at such periods, better tailoring subsequent groups.

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
```

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**.

References

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

[2] Gerdes A., Heeren B., Jeuring J.: Teachers and students in charge, In: 7th European Conference on Technology Enhanced Learning, LNCS, Vol. 7563, Springer Verlag, 2012, pp 383-388.

[3] Andrew S. Glassner, An introduction o ray tracing, Academic Press Ltd., London, UK, 1989

[4] 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.

[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.: The 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.

[9] 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.

[10] 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.