$(\text{Easy} + \text{Hard})/2 =$ balanced advanced rendering assignments

R. Kuffner dos Anjos [1] and M. Billeter[1]

[1]University of Leeds

Abstract

Computer graphics is a topic that incorporates high-performance programming, advanced mathematics, and knowledge of physics. Typically taught at later years of university education, the expectation is to deliver advanced concepts, nearing state of the art technology. However, with the need to assess a wide range of aspects, designing an assessment strategy that encourages high achievers while allowing success for people struggling with basic concepts usually ends favoring one of the groups, leading to high amounts of failure, or compromises in terms of covered content and challenge. In this paper we present a practical programming assignment strategy to cater to students with varied background knowledge, allowing for extensive feedback, evaluation of highly advanced concepts, and a manageable workload. Our approach was validated in three different cohorts over three years, resulting in positive feedback, better performance, and better coverage of the material.

CCS Concepts

• ***Computing methodologies*** → ***Computer graphics;*** • ***Applied computing*** → ***Education;***

1. Introduction

Assessing student knowledge in advanced computer graphics courses effectively is a challenging task [SWLR17]. Computer graphics often covers a wide range of topics [BWF18], ranging from mathematics and physics to hands-on software engineering and utilizing specific APIs.

As an advanced course, and not an introductory course to computer graphics, we have the desire to push the boundary of what is taught to prepare students with advanced skills to join an increasingly competitive job market. We have therefore chosen to focus on practical exercises and assignments, where students implement several different aspects and methods in rendering. This implementation-heavy approach brings several challenges. We have to deal with large variations in prior knowledge. The focus on practical work can bring a heavy workload to students, further amplifying differences. The core challenge has been to organize our work and assessment to help students cope with the workload, help with planning, all without compromising on the content that we want to cover and assess.

Herein, we report our findings in the context of the masters course, *Advanced Rendering*, taught in our Higher Education Institution. In this paper, we describe the assessment strategy introduced over the past three years with the following goals:

- cover theoretical and practical elements at different levels
- require familiarity with a core selection of essential topics
- more opportunities for feedback and problem-based learning
- a balanced level of challenge and resulting grade profile
- manage use of generative AI in completing assignments.

2. Previous work and motivation

In this sections we describe the different challenges we tackle in our assessment design, and the related literature to each one.

Breadth: Computer graphics requires integrating different areas of knowledge such as mathematics, physics, programming and spatial reasoning. [BWF18]. Raytracing is a good example, commonly considered one of the most challenging aspects of many computer graphics courses. Previous research [LWLR*25] has suggested further integration of practical exercises and strengthening students fundamental knowledge through visualization and practical examples. This finding is in line with previous research which favours teaching computer graphics courses "in a context" [Cun08, APH*12], meaning with an application in mind. This has been shown to increase both student motivation and understanding of certain concepts, such as transformations and rendering. In fact, practical assignments help further establish their spatial reasoning capabilities through visualization of the mathematics students are being taught [RHLL20]. This relationship between spatial skills and performance in computer graphics courses has been demonstrated by Resnick et al. [LWLR22], which motivates considering theoretical concepts alongside practical exercises as a good assessment method for computer Graphics.

Marking: While successful experiences on automated marking have been reported by previous work [HWD*25, WCS*18], the type of questions supported tend to be limited to pipeline states and parameters, or pixel-wise comparisons which may not be accurately representative of how correct or incorrect a rendering process is. While relevant in a large undergraduate cohort, we describe our

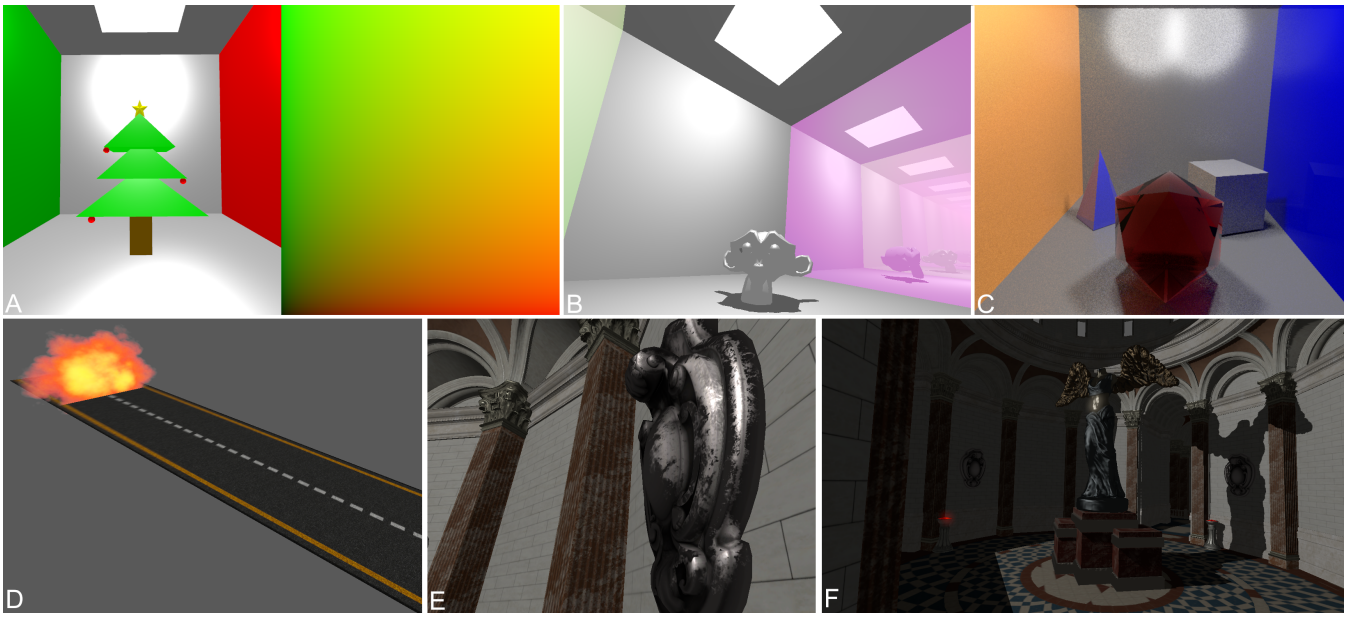

Figure 1: *Different stages of the practical tasks in our course. From left to right: state after self-guided tutorials, basic tasks (Assignment 1) and advanced tasks (Assignment 2). Top: ray tracing components. Bottom: rasterization components.*

efforts for a masters level course with 20-40 students. Another limitation of automated marking is providing meaningful, personalized feedback to each learner.

Support: We would like to introduce a problem based learning (PBL) [YG16] approach integrated with the assessment so students can work through problems on their own. PBL has gained popularity recently promoting better knowledge retention [BBA09] even with little guidance. However, critics of this method point to the need of excessive scaffolding to prevent students from failing, which in consequence decreases how advanced the taught content can be [SKC07]. We believe this can be addressed by providing scaffolding on basic content, but removing it for advanced tasks.

API: While OpenGL remains a popular API for learning, the games industry especially is often working with more modern APIs, including Direct3D and Vulkan. Fortunately, these modern APIs share many concepts. No-API approaches have been proposed [Gei25], and these have found success in introductory courses. However, in preparing students to the job market, familiarity with the concepts shared by modern graphics APIs is a significant advantage. Unterguggenberger et al. [UKW23] describe their efforts in teaching a course split between Vulkan and OpenGL, with split results on performance. They demonstrate viability of teaching the Vulkan API, albeit using a custom abstractions for a more gentle introduction. While we believe OpenGL can still be used as an introductory API thanks to its relative simplicity, reasonable support in many devices, we also focus heavily on the more up-to-date Vulkan API for our MSc-level course. Nevertheless, (post-)modern OpenGL shares some concepts with the more modern APIs, making it possible to use it to introduce concepts like shaders and data management (textures, buffers) early.

3. Assessment strategy

In our advanced courses, we have focused on assessment through practical assignments. Our goal is to give students an opportunity to practice graphics software engineering, as this is very likely a central component in their early career. We want to pose real-world problems in a problem-based learning approach, where there is ample of opportunity for feedback and support during their work.

A core shift in our approach is split the assessment by level of challenge, instead of by theme. Previously, we organized the teaching by topics. For example, in the rendering course, we cover ray tracing and rasterization as two central topics. We split each topic into multiple small assignments covering different aspects of the topic, with each assignment spanning tasks with increasing difficulties. This is quite a common approach [LWLR*25].

We observed a few challenges with this approach in our student cohorts. One challenge related to the timing of assignments with the delivery of teaching. Assignments could only be handed out once significant portions of the content were covered, especially for more advanced aspects of the material. In practice, this reduced the time left for the practical work. Advanced problems would naturally take more time to complete. We found that students would often attempt to complete every task on a given assignment, including very advanced ones, to the detriment of balancing the workload between different courses or even subsequent assignments. The focus on early advanced tasks would also encourage skipping later, potentially more fundamental content. The approach left less time to provide feedback on fundamental concepts (e.g., such that were part of later assignments).

Course structure: To provide context, we will first outline the practical structure. Some limits, e.g. course duration, are often im-

posed externally and can be difficult to influence for individual courses.

Our course runs over 8 weeks, with 5 hours of lectures and 6 hours of lab time scheduled each week. We have organized the assessment into two practical assignments. The first two weeks are dedicated to foundational concepts, introducing different rendering approaches (ray tracing with path tracing, and the rasterization pipeline). This is followed by two weeks focusing on modern rendering APIs and hardware, and specifically the Vulkan API. The remaining four weeks are dedicated to a core selection of modern rendering techniques and advanced methods. We supplement lectured content with practical (non-assessed) exercises/tutorials. All practical content -exercises and assignments- is handed out in the first day of teaching to support student planning. By Week 4, all the taught content required for the first assignment will have been covered. Students are expect to be able to complete it from then on.

3.1. Assignment 1: Pass/Fail

The first assignment is a pass/fail assignment. It must be completed fully by each student to receive a passing grade for the course. Completing it will also always award a 50% grade, which is the minimum required to pass a course at our institution. The assignment is split into a ray tracing and a rasterization component.

Given the usual coverage of topics in computer graphics courses [BWF18], we consider these to be the concepts a student should grasp to be approved in an "advanced course".

Ray tracing: Students implement a Whitted-style ray tracer with direct illumination calculation, secondary rays for shadow testing, and a recursive approach for reflections (1B). This is implemented over an OpenGL framework where students complete a rasterized reference renderer in a self-guided tutorial (1A), which includes a simple pipeline to display the ray traced results (1A, right). The ray tracer itself is implemented on the CPU, optionally accelerated with OpenMP. The focus is to ensure understanding of transformations, coordinate spaces, shading; to demonstrate the structure of a simple ray tracer; and begin exposing students to some of the core rendering problems. The OpenGL tutorial works as an early introduction to some of the rasterization concepts that will be revisited in depth in the later part (e.g., shaders, resource management, plus standard concepts like projections and depth testing). This component can be completed as early as the second week of the course.

Rasterization: Students implement real-time renderer using the Vulkan API, with 3D scene navigation, (Physically Based Redering) PBR shading, some debug visualizations. They use multiple rendering pipelines and implement a simple post processing effect using a render-to-texture approach (1E). These components follow a Vulkan tutorial with five self-guided exercises to familiarize students with Vulkan code (1D). The goal is to introduce students to a modern graphics API and deepen their experience with shader programming. Further, it exposes them to standard PBR shading and common tasks like render-to-texture. The assignment also includes a specific part on debugging graphics, where students implement simple visualizations rendering properties and intermediate values.

Students are able to complete this component by Week 5.

Base code: For each of the two parts, we provide a basic project setup. It includes code and third party libraries required to complete the task. This includes minimally window management, classes for "graphics math" (linear algebra, quaternions, ...), as well as some helpers for loading 3D models and materials. The ray tracing base code then further includes code for camera control and scene representation. The rasterization base code further includes Vulkan components (Vulkan headers, a dynamic Vulkan meta loader, and shader compilation tools).

The base code is built on the code from the self-guided exercises, meaning students build up familiarity with it through our practical tutorials.

Assessment: We assess the first assignment in-person, directly in the shared labs, via an oral Q&A process. Staff and teaching assistants are present in the lab sessions to provide support. Students can at any time request to demonstrate their work. While less common, it is possible for them to attempt to demonstrate only a subset of the tasks, before moving on to others. Students will demonstrate their work by running their programs and showing the results. When deemed sufficiently error-free, students will be asked questions. They must be able to confidently explain their code, answer theoretical questions, be able to implement on-the-spot modification requests, and overall demonstrate an understanding of both the concepts and their code. Failing any of this does not typically result in an overall failure: rather, students are asked to fix problems or study further and then request another Q&A opportunity. Staff keep notes via a simple web application to track each student's status and to easily share information across sessions and staff members. We also use the Q&A process to identify and raise concerns about academic integrity early.

3.2. Assignment 2: Graded

After completing the first assignment, students can attempt to achieve a higher grade with an additional set of advanced tasks. At this point, students are certain that they have passed the course. The decision to attempt any of the tasks is therefore a choice by the student, according to their workload and ambitions. Students are able to start with these task as soon as they finish the first assignment. The second assignment builds on the code from the first one (and, consequently, uses the same base code). We will only consider advanced tasks if the first assignment is completed in time. The second assignment extends slightly beyond the end of the course, into what is normally an exam period. This gives students extra time on these tasks and simplifies course planning significantly. Assignment 2 has a similar split between topics as Assignment 1. We update these tasks frequently to keep the course current with current practice. The current set of tasks is outlined in the following.

Ray tracing: Advanced tasks include introducing transparency and refraction and the Fresnell effect. Students are also asked to transform their implementation into a path tracer with next event estimation (NEE). This also covers handling of anti-aliasing, area lights, light in media, and the correct integration of NEE with transparent objects to produce caustics (1C).

Rasterization: Students are given a choice to implement methods including deferred rendering, normal mapping with compres-

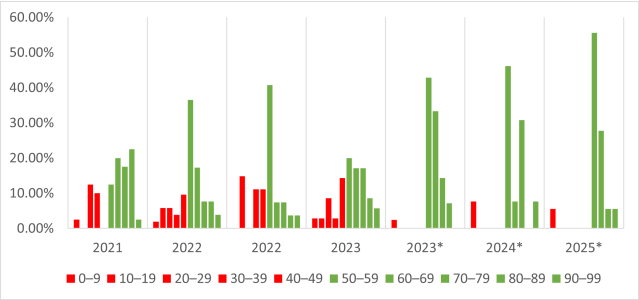

Figure 2: *An overview of the grade distribution for MSc graphics courses in the past years. Years with a star (*) mark iterations of the course(s) that utilized the described assessment approach.*

sion, shadow mapping and bloom on top of their existing renderer (1F). In addition, the tasks include more advanced diagnostics visualization, to explore aspects like overdraw, overshading and rendered mesh density.

Assessment: Assignment 2 is assessed offline. Students submit their code with a report. We ask students to describe their work, evaluate their results and analyze critically. Certain tasks also include specific non-programming questions that require further analysis. Through their observations, analysis and discussions in the report, students can demonstrate a deeper understanding in addition to the ability to practically implement the aforementioned methods.

4. Results and conclusion

Looking back at the problems and challenges from 1, we believe this design successfully addresses those. By separating assessment into essential and advanced components, we are better guiding our cohort towards meeting the learning objectives of the course, representing the minimum bar for passing (see Figure 2). Previously, students would spend too much time trying to tackle advanced problems in an assignment, and perhaps struggling to complete these satisfactorily, instead of working on other more fundamental concepts. This would increase the likelihood that they would perform poorly. Moreover, some students could pass the course by being an expert on only one part of it (e.g., focus entirely on ray tracing, ignoring rasterization entirely). Our current approach mitigates these problems.

We found that examining Assignment 1 was manageable with our class sizes for a masters course (typically between 20-30 students). In practice, the assignment would require two Q&As, one for each component, lasting around 10-15 minutes on average. One member of staff could handle Q&As most of the time; close to the final deadline, we employed additional help to make sure everybody had a fair chance of finishing in time. For larger classes, we believe automated grading [HWD*25, WCS*18] could be used to streamline the process. For example, it could be used to validate the code, so the staff focus on the oral examination.

The main challenge we found is keeping students on top of work regularly. For this approach to be successful, guiding students towards completing the minimum objectives swiftly is key. We encountered some challenges with concurrent courses, which would pose a combination of basic and advanced tasks at the same time as our first assignment. This re-created the situation we were trying to avoid. We will focus more on directing the students proactively during the semester and attempting to reorganize tasks to mitigate clashes with demanding tasks from other courses.

References

[APH*12] ANDERSON E. F., PETERS C. E., HALLORAN J., EVERY P., SHUTTLEWORTH J., LIAROKAPIS F., LANE R., RICHARDS M.: In at the Deep End: An activity-led introduction to first year creative computing. *Computer Graphics Forum 31*, 6 (2012). doi:10.1111/j.1467-8659.2012.03066.x. 1

[BBA09] BRUNSTEIN A., BETTS S., ANDERSON J. R.: Practice enables successful learning under minimal guidance. *Journal of Educational Psychology 101*, 4 (2009). doi:10.1037/a0016656. 2

[BWF18] BALREIRA D. G., WALTER M., FELLNER D. W.: A survey of the contents in introductory computer graphics courses. *Computers & Graphics 77* (2018). 1, 3

[Cun08] CUNNINGHAM S.: Computer graphics in context: an approach to a first course in computer graphics. In *ACM SIGGRAPH ASIA 2008 Educators Programme* (2008), SIGGRAPH Asia '08, Association for Computing Machinery. doi:10.1145/1507713.1507715. 1

[Gei25] GEIGEL J.: A no-API approach to an introductory computer graphics course. doi:10.2312/eged.20251009. 2

[HWD*25] HOOPER S., WÜNSCHE B. C., DENNY P., LUXTON-REILLY A., KONINGS N., CAMPBELL A. D.: Educator experiences with automated marking of programming assessments in a computer graphics-based design course. In *Proceedings of the ACM Technical Symposium on Computer Science Education* (2025). 1, 4

[LWLR22] LIU K., WÜNSCHE B. C., LUXTON-REILLY A.: Relationship between spatial skills and performance in introductory computer graphics. In *Proceedings of the 27th ACM Conference on Innovation and Technology in Computer Science Education* (2022). 1

[LWLR*25] LIU E., WÜNSCHE B. C., LUXTON-REILLY A., LANGE-NAWKA D., HOOPER S., THOMPSON S. E.: Tracing brilliance: Analysing student performance in ray tracing and problem-solving capabilities and approaches. doi:10.2312/eged.20251007. 1, 2

[RHLL20] RESNICK I., HARRIS D., LOGAN T., LOWRIE T.: The relation between mathematics achievement and spatial reasoning. *Mathematics Education Research Journal 32*, 2 (2020). 1

[SKC07] SWELLER J., KIRSCHNER P. A., CLARK R. E.: Why minimally guided teaching techniques do not work: A reply to commentaries. *Educational psychologist 42*, 2 (2007). 2

[SWLR17] SUSELO T., WÜNSCHE B., LUXTON-REILLY A.: The journey to improve teaching computer graphics: A systematic review. In *International Conference on Computers in Education* (2017). 1

[UKW23] UNTERGUGGENBERGER J., KERBL B., WIMMER M.: Vulkan all the way: Transitioning to a modern low-level graphics API in academia. *Computers & Graphics 111* (2023). doi:10.1016/j.cag.2023.02.001. 2

[WCS*18] WÜNSCHE B. C., CHEN Z., SHAW L., SUSELO T., LEUNG K.-C., DIMALEN D., MARK W. V. D., LUXTON-REILLY A., LOBB R.: Automatic assessment of OpenGL computer graphics assignments. In *Proceedings of the Annual ACM Conference on Innovation and Technology in Computer Science Education* (2018). 1, 4

[YG16] YEW E. H., GOH K.: Problem-based learning: An overview of its process and impact on learning. *Health Professions Education 2*, 2 (2016). doi:10.1016/j.hpe.2016.01.004. 2