Introduction to computer graphics: a visual interactive approach

C.Loscos

IUT of Reims, LICIIS laboratory, University of Reims Champagne-Ardenne, France

Abstract

Computer graphics is a difficult topic, requiring associating mathematics and programming skills. When initially taught at undergraduate levels, there are several factors which discourage students. First, programming a first computer graphics program requires a substantial initial framework which can be intimidating for many of them. Second, understanding and applying mathematical concepts is very often overwhelming.

To counter this intimidating feeling, a new teaching approach was proposed in 2018 to 3rd year undergraduate computer science students. The course was split into two parts, theory and practice. The theoretical concepts were seen in class, with course handouts and table exercises resembling closely to traditional computer graphics learning. The originality of the course comes from a new way of practicing 3D programming. Practical labs were built upon the Unity game engine programming platform, adapted to match the theoretical concepts seen in classroom.

Conclusions are drawn over 4 years of teaching this course. When taught using an accompanying easy-to-access graphics programming platform, computer graphics becomes a more attractive course for students with lower mathematics and programming skills. It is also very satisfactory for skillful students as it enables them to grab and master concepts quickly to reach interesting final lab achievements.

CCS Concepts

• ***Computing methodologies*** → ***Computer graphics;*** • ***Applied computing*** → ***Interactive learning environments;***

1. Introduction

Computer graphics is a difficult topic, both to teach and to practice. It requires skills both in mathematics and computer science, with quite advanced programming competences. It is often taught only from the graduate level, when students have validated strong competences in advanced programming. Introducing computer graphics at earlier learning stages, like at undergraduate levels, requires adapting in order to keep motivation and attractiveness. Otherwise, students may rapidly feel overwhelmed and discouraged.

In this paper, we propose a new teaching methodology which builds on Unity [Uni], a pre-existing graphics programming platform, a game engine development framework. There exist many online tutorials to learn to use Unity. They are dedicated to specific tasks and need pre-knowledge of computer graphics. Far from being a dedicated tutorial to the Unity platform, the pedagogical approach of the course makes use of the offered facility while introducing graphics concepts with immediate visual feedback and interactive manipulation.

1.1. Motivation

While mathematics prerequisite may first feel accessible, today's students who reach computer science do so often because they do not want to pursue a degree in mathematics. Addressing mathematically concepts in classroom often leads to disinterest.

Computer graphics programming requires most of the time to master iterative and recursive algorithms, and object-oriented languages. Addressing graphics libraries and accelerated programming together with basic concepts would necessitate an extremely well-defined pedagogical method to keep the students' full attention. One may even consider the difficulty of approaching current graphics shader programming, which can be very abstract for computer scientists in their early learning years. Adding mathematical concepts in the programming content results often in the automatic copy of line coding without the real understanding of concepts.

A possible approach is to build the whole graphics framework or by providing a first framework skeleton to students. Reaching first programming steps after a substantial initial framework development may leave an overwhelming feeling to students and the impression that the topic is out of reach.

The course is designed to deliver basic computer graphics concepts in very few hours (30h in total) to 3rd year undergraduate students. This course is one of the two optional classes students need to pick out of three. Most students chose it by default without any pre-knowledge of the field nor the intention of pursuing it. At the end of the course, we expect that: (1) All students are capa-

DOI: 10.2312/eged.20221039

ble of describing the basic concepts of computer graphics, and they should understand what it takes to pursue in this area; (2) Students understand that even if some mathematics are necessary in computer graphics, they are capable of grabbing concepts quickly and they can pursue in this area.

1.2. Originality

To fulfill our objectives and to counter the above-described intimidating feeling, a new teaching approach have been proposed to students since 2018. The course is split into two parts: theory and practice. The theoretical concepts are seen in classroom, linking in a classical manner lectures and table exercises. The originality of the course comes from a new way of practicing 3D programming. Practical labs are built upon the Unity game engine [Uni], adapted to match with the theoretical concepts seen in lectures. It is not a course to learn to program the Unity game engine but rather a roundabout way to use Unity and its programming interface to teach computer graphics.

The choices made for this course partly follow the constructive approach to learning [BO15] [Sim93]. The teacher is there to provide a theoretical background which students can then learn through manipulations on the unity game engine. While following a few guidelines, they can make personal choices thus going to very subjective add-ons. The swap between theoretical and practical learning session could also be seen as active learning [FB09] although this is not done within a same session. The way the practical labs build up can also be seen as project-based learning.

1.3. Paper overview

In the following, we first review other approaches to introductory computer graphics programming in section 2. We detail the course scope, objectives, and public in section 3. We describe the course setup and delivery conditions in section 4. We give details on the Unity-based practical exercises in section 5. Finally, we give examples of achievements and evaluate the success of the course in section 6, before concluding in section 7.

2. Positioning against other teaching approaches

Teaching introduction to computer graphics have always needed adjusting topic focus and programming support. It was surveyed in [BWF17] where different methodologies were outlined, and topics like graphics pipeline, rasterization, lighting, etc., were ranked by most taught in introductory courses. This survey also depicts the choice of programming languages. Unlike most teaching methods, our aim is to address the full computer graphics pipeline, and its main steps (modeling, navigating, animating, and lighting).

Most of the time, introduction to computer graphics is set with a consequent teaching hour range (often at least 60h) and targets master students. However, teaching graphics programming starting from hardware understanding has been challenged in many ways [CXR18]. While it could be acknowledged that starting with ray tracing gives a nice insight to 3D programming [CXR18], some courses have based programming efforts on using already existing platforms, like [AG16], or by providing a purposely built programming platform like [BSP17]. Others have adapted to WebGL and ThreeJS, like [AB15]. The course described in this paper is set only to be taught in few hours (10h of practical labs) while most classes previously addressed targets 60h to 100h. When setting up the class, we considered several choices (ThreeJS, introduction to OpenGL, own teaching platform, ray tracing). However, our choice was to go visual and interactive like offered by the Unity game engine development platform. The Unity game engine has already proven to be a successful platform to teach computer game programming to students [Dic15].

When teaching to undergraduate students, similar pedagogical approaches to master levels are often implemented. Designing an introductory course [DC17] for first year students is even more challenging, and simplification associated to the programming language (here java), and 2D formulations can be made. It is more and more common to face undergraduate computer science students with low confidence, low performance, or low interest in mathematics. Shesh [She15] explains that although students may be reluctant to maths, they still show interest to computer graphics. He proposed one way to teach students both maths and programming while keeping their interest. Our decision was to use the Unity interface to display polygon coordinates, coordinate systems, transformations, material properties, etc. and have it accessible through easy interactive manipulation.

Finally, the sanitary pandemic situation brought significant teaching challenges the last two years, with remote teaching becoming compulsory to many of us. Several solutions were proposed, as in [RMV*21]. We show in section 6 that the practical lab setup well adapted to remote teaching.

3. Course scope, objectives and public

3.1. Context and public

In our university, the computer science undergraduate program offers, in the last semester of the 3rd year syllabus, the choice between three options: Introduction to computer security, Introduction to digital imaging, Introduction to artificial intelligence. This aims at giving an incentive to students to open up their possible application to the three master programs offered in our university. As it is an introductory class, the total teaching volume is only of 30h.

The third-year undergraduate students follow a computer science program. They are every year around 40 students who enroll the introduction to digital imaging option, the course discussed in this paper. They have a shallow knowledge in mathematics and have good programming and algorithmic background. They are taught Java and C programming languages. One can note that Unity scripting is done in C#, language not taught to our students. Some students integrate the 3rd year program after successfully obtaining a 2-year technological degree, and may have learnt other languages (Python, C++, C#). These students are particularly reluctant to mathematical equations and concepts.

3.2. Course scope and objectives

The course was designed to reach the two following learning objectives at the end of the course. First, students should have un-

derstood the different components of a 3D software. Second, they should know how the different graphic elements can be manipulated to reach an interactive 3D content.

At the end of the course, we expect students to reach the following competences:

1. To be able to define the different 3D components: scene modeling, interaction and navigation, rendering, and simulation.
2. To know the properties to program each 3D graphics component: model structures, dynamic scene and camera motion, physical object interactions.
3. To understand and be able to program in a 3D graphics game engine like Unity [Uni].

More generally, the course should open to programming competences to bring students to more autonomy, typically such as:

1. To quickly adapt to a new programming language (here C#).
2. To know to program scripts and short programming components in existing software.
3. To learn to use programming game engines.

4. Course syllabus and setup

4.1. Course syllabus

The course syllabus is as follows:

1. Elements composing a 3D scene for 3D imaging.
2. Useful maths
3. Scene and objects:
   - What is a 3D scene?
   - What are the components of 3D objects: points, lines, polygons, polyhedrons, normals.
   - How to build 3D content: topology, volumes, surfaces, scene graph.
4. Camera models: 3D coordinate systems (scene and camera), transformations.
5. Rendering:
   - The local illumination equation and the different types of reflections.
   - Shadow properties and algorithms.
   - Rasterizing, texturing.
6. Animation:
   - Animating cameras.
   - 3D interaction between objects.
   - Physical properties (collision, gravity).

4.2. Teaching modalities

The course is taught in three types of classrooms: 10h of lecturing, 10h of table exercises, 10h of practical labs. Each are given every week by steps of 2h, thus students have each week 6h in this course on average (see table 1). All students are in a single classroom for lectures and table exercises, but they are spit into smaller groups for practical labs (between two or three groups depending on the year). When possible, lecturing is given upfront, to leave a week between practical exercises and lecturing. The course attendance is compulsory only for the table exercises and the practical labs. Students can choose to install Unity on their own laptop and come in the classroom with their personal laptop, while Unity is also installed on University lab computers.

The course is accessible on the moodle server of the university to which students are automatically registered. There, students find the lecture hand notes and the exercise statements. Lectures and table exercises are based and adapted to teaching content from the book [SSC02]. On the moodle website, there is a special section for practical labs. At each session, students access a lab session form, which is composed of a description of the work to do and a section to fill in directly in the form. They can enter text, screen shots, and code. Practical labs are performed by pairs. Lab sessions are set so that they can be accomplished each within the 2h dedicated time. Students may continue this work outside the lab sessions. However, personal work time is planned to be dedicated to theoretical learning and finalizing lab work.

The evaluation is twofold. Students are evaluated on the theoretical part via a written test (2h) which counts for 60% of the overall grade. Reports of practical labs are marked per pair of students and count for 40% of the overall grade.

It should be noted that during the 2020/2021 university year, the course was given fully remotely and taught using the Microsoft Teams video conferencing platform [Mic].

4.3. Cross-pedagogical links

The course follows the syllabus described in section 4.1. Lectures, table exercises, and lab sessions are set to synchronize learning in this order. Each concept is first seen in lectures, worked on table exercises, and then experimented in practical labs. Table 1 shows how learning topics are synchronized.

Week	Lectures	Exercises	Labs
Preamble			Unity install
Week 1 (2h/2h/)	Introduction Mathematical basics Planes, Polygons	Mathematical basics Polygons	
Week 2 (2h/2h/2h)	Data structures Scene graphs Camera models	Polyhedrons data structures Scene graphs Camera transforms	Unity discovery
Week 3 (2h/2h/2h)	Local lighting Shadows	Local lighting Shadows	Scene modeling Complex objects
Week 4 (2h/2h/2h)	Rasterization Gouraud/Phong shading	Rasterization Gouraud/Phong shading	Camera settings and scene navigation
Week 5 (2h/2h/2h)	Recap	Recap	Lighting and shading
Week 6 (/ /2h)			Animation Gaming

Table 1: *Per-week synchronization of topics.*

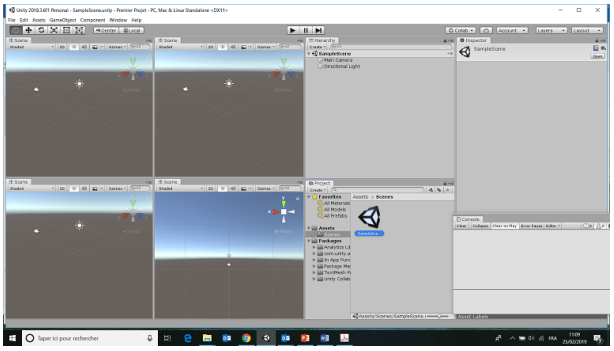
5. Learning steps

In this section, we give details of the teaching labs based on Unity programming. Each following section corresponds to a 2h lab session, to be done in this order. Each lab builds on the results of the previous one. Each lab description is a step-by-step exercise statement which leads the students towards a final solution. Students provide intermediate answers to questions and upload screen captures and programming code files at the different steps when appropriate and asked.

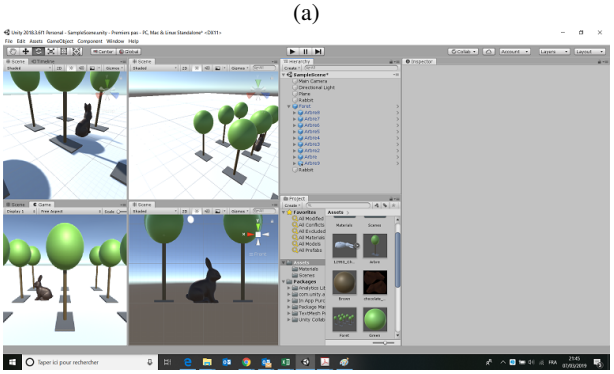
5.1. Introduction to Unity and first modeling

The objective of the first lab session is to globally grab the programming environment. The aim is for all students to get the organization of the Unity programming platform and to identify the 3D graphics elements as seen in lectures. In particular, they need to identify how coordinate systems are in place, and check for cameras and light sources. They also need to make the difference between the editing and the running modes. The basic Unity window development system is shown in figure 1(a).

They are then asked to add simple objects to the scene, including loading a more complex object (here a bunny) from a file. They can identify how to replicate objects (prefab). An example is shown in figure 1(b).



(a)



(b)

Figure 1: *(a) The Unity programming platform before any programming. (b) After adding a prefab (tree), replicate it, and adding a rabbit model from a file.*

5.2. Scene graphs and physical interactions

In a second lab session, students are asked to replicate an articulated object (we call it "robot") as seen in the second week lecture. Whereas Unity does not have scene graphs per say, it is possible to build a complex object made of simple objects which all have a relative position one-to-another, with a hierarchical description (see an example in figure 2(a)). Students need to identify how a local coordinate system is associated to each object and that it is possible to compute the necessary rotation/scale/translation. While Unity offers the possibility to place objects using mouse interaction, it is specifically required that students give precisely the object position relatively to the previous one, by hand-typing values. Therefore, they need to apply in a practical way the mathematical concepts seen in lectures and classroom tutorials. This also helps them understand the difference between the global scene coordinate system and the local coordinate systems. At an intermediate step, they upload both a picture of their final hierarchical model of the robot and the set of parameters entered manually in the interface. The final expected results is shown in figure 2(b).

In a second part, they add interaction and physical properties. They enable collision and gravity, and start programming in C# to animate the robot. They are asked to move the robot on a plane, pushing a small cube, which changes color when reaching locations identified by squares. In this manner, they learn scripting and grab notions of spatial motion programming. They also know to change states of objects (in this case, position and color). An example is seen figure 3. The small cube changes color when entering a colored squared area. At the end of this step, they are asked to upload screen captures and their C# code files.

5.3. Interactive viewing and navigation

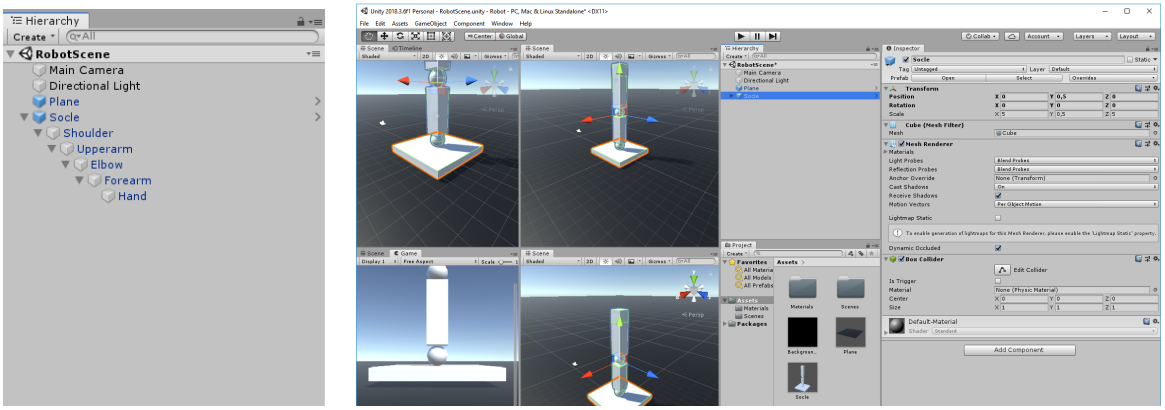
In a third lab session, students are asked to control three cameras. One is a first-person camera attached to the robot, one is a camera that follows the robot at a certain distance, and the last camera always points at the cube. An example of camera transform interface is shown in figure 4. They need to interact with the camera controls, enabling to switch cameras, and they also need to attach a C# script to each camera when the robot or the cube change position. They can then explore the "look at" profile of a camera and explore the different rotation/translation modes of navigation.

5.4. Lighting simulation

The fourth lab session is dedicated to lighting (see figure 5). In this session, students explore different types of light sources (point, polygonal, directional, spot). They can edit object materials and add textures. They attach a spotlight to the robot to illuminate in the direction of sight. They also explore the different types of shadow (hard, soft). This is also supported by c# scripting.

5.5. Going further - interaction and gaming

In their last lab, students' creation is let out. They are asked to use the learnt concepts to design an interactive game scenario. For the most advanced students, this was already started at previous lab sessions if they had finished the required tasks before the end of the



(a) (b)

Figure 2: *(a) An example of hierarchy in Unity. (b) Coordinates of each node object of the hierarchy with its local coordinate system.*

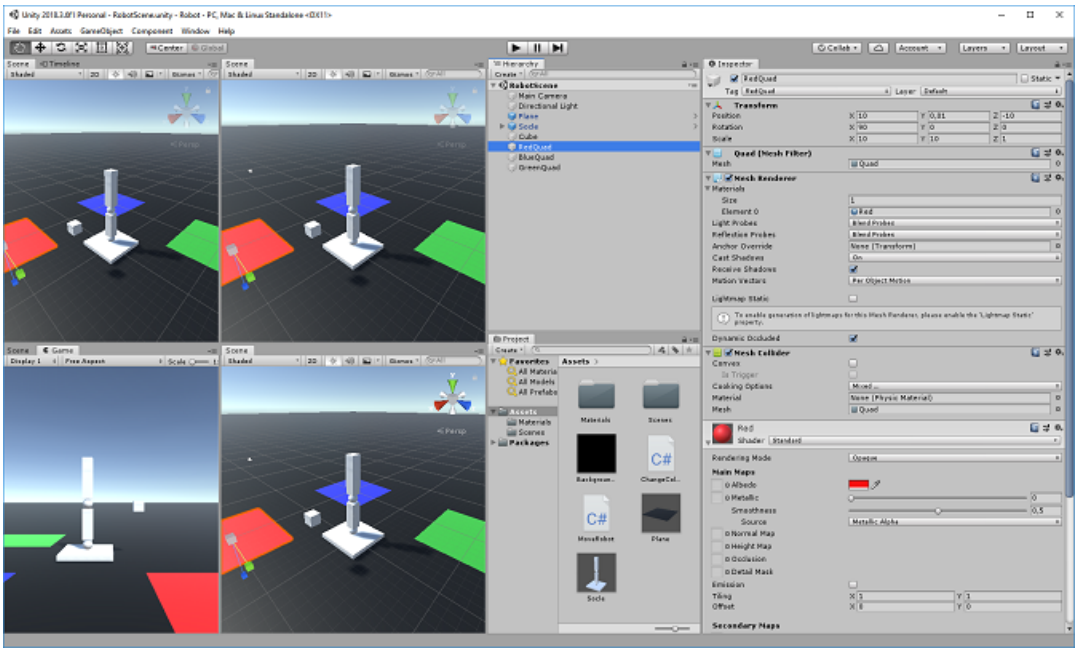


Figure 3: *Example of setup with the robot, the cube, and the colored square areas.*

session. Here, they can explore the addition of objects, lighting, interactions, display score, and implement targeted actions. Example of productions are presented in section 6.

6. Learning quality analysis

6.1. Students' productions

Most students who worked regularly achieved the first four lab sessions. About 25% of students uploaded a game simulation. Examples of student achievements are shown in figure 6. Here we can see that some students use the elements already programmed to upgrade it to a game. Some others worked on improving appearance. A few of them used what they have learnt in order to build a completely new world, loading complex objects. Some game involved reaching a target, others making objects fall, some adding a timer. Additional results (video produced by the students) are shown in the supplementary material.

6.2. Students' results

Most student succeeded, passed this class and graduated. It is clear that what they preferred was the Unity programming labs. However, keeping in mind the reward of programming in Unity, it was

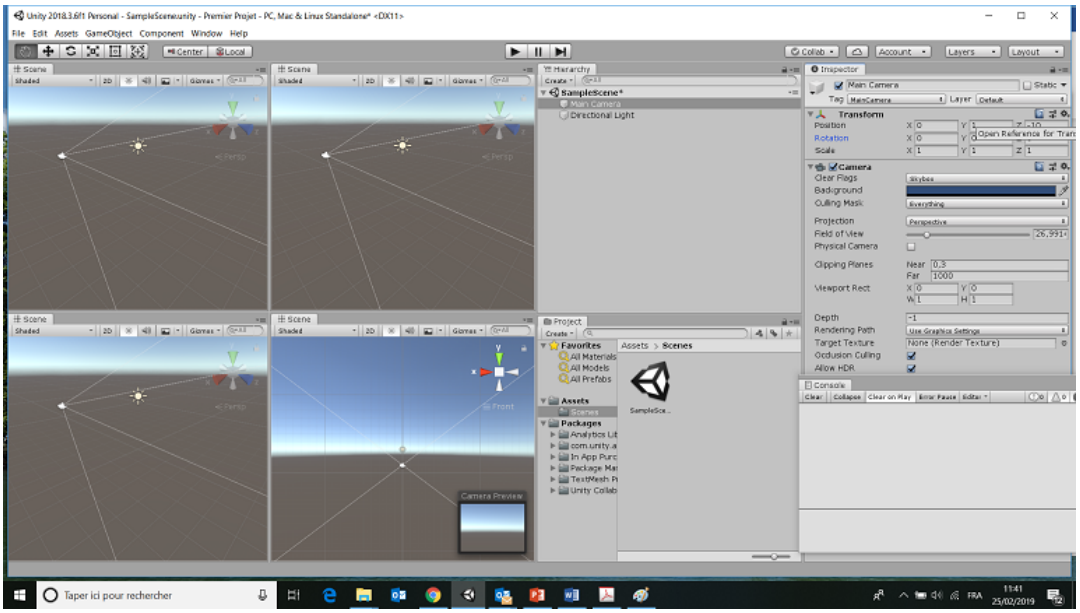


Figure 4: *Example of camera setup.*

possible to get their attention for the table exercise sessions. Table 2 summarises their results, showing the min, max and average grades. Every year, only a single pair of students did not validate the practical labs. Applying the weighted average on practical labs and theory results, only around 25% did not validate the course. However, half of them had a grade which allowed them to compensate with other courses to validate their overall degree, and the other half did not pass the degree and needed to retake the year.

It is to be noted that year 2020/2021 was taught fully remotely due to the sanitary conditions. We can notice no effect on the practical labs, and even, grades are higher. Students all learnt with a strong motivation and probably worked outside lab sessions. However, theoretical sessions were not as effective and there is a strong drop of grade performance for the written exam.

Year	Theory			Practice		
	Top	Lowest	average	Top	Lowest	Average
2018-2019	17.6	3.1	10.8	20	8.5	14.2
2019-2020	19.75	2.5	9.9	18	5	16
2020-2021	20	0.5	7	20	3.5	16.3

Table 2: *Average grades over the last three university years.*

6.3. Overall conclusions

After teaching this course for 4 years (3 years evaluated), our conclusion is that the offered pedagogical approach for lab sessions enabled computer graphics to become a more attractive class for students with lower mathematics and programming skills (enrollment increased this year, with 48 students registered). It is also very satisfactory for students with shallow skills as it enables them to grab and master concepts quickly to reach interesting final lab projects. They often give spontaneous positive feedback on the Unity-based labs. Although often questioned in the first lecture, none of them complain about having to program with C# when getting to practical labs. Every year, several students decide to pursue computer graphics related master's degrees.

It is not compulsory to attend lectures. Since handouts are provided on the course moodle website, some students decide not to attend the lectures. This clearly affects the learning effect of the theoretical background since the first time they discover concepts is through the table exercise class. Unfortunately, with the university regulation in place, it is not possible to make the lecturing sessions compulsory.

7. Conclusions

This paper presents a new way of teaching practical programming of computer graphics to get away from heavy framework programming and mathematical complexity. The course was designed to use the Unity game engine to illustrate computer graphics concepts and help students grabbing quickly a first insight, in only 10h of practical labs (30h of teaching in total).

The course is designed to introduce computer graphics to undergraduate students who have no previous knowledge of computer graphics and shallow mathematical skills. Many of them take this course by default out of two other options, without any solid motivation. Obviously, the overall concept could be adapted to longer project-type classes or to other types of graphics game engine.

Results show that the course is attractive to students and that most of them reach a basic computer graphics level. The course helps them outlining the programming and mathematical skills necessary to pursue further studies in master and to target a career in this area.

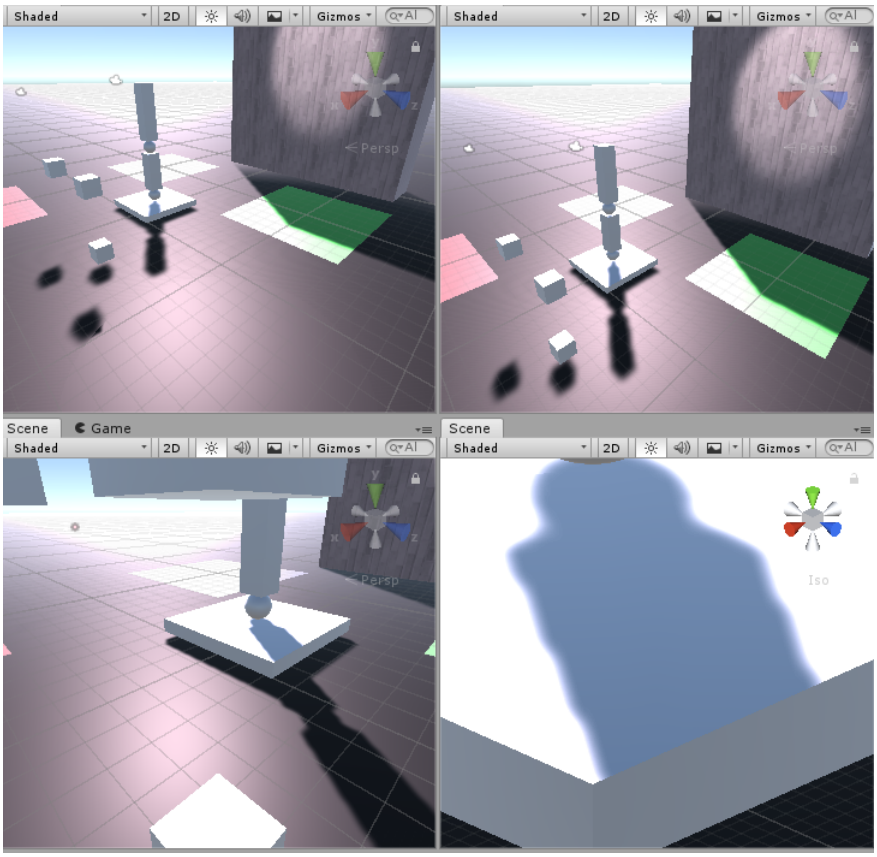


Figure 5: *An example of lighting, shadows, and texturing.*

This paper is a first study. It would be interesting to pursue this with other types of validations. For example, it could be taught in parallel of a more traditional class (i.e., OpenGL) for comparison. A reviewer also suggested to swap the order of practical labs and theoretical sessions, thus engaging learning more through inverse classes.

8. Ackowledgement

This work is the result of teaching with the mathematics and computer science departement of the UFR exact and natural sciences of the University of Reims Champagne-Ardenne. I would like to thank Jessica Jonquet, Théo Barrios, and Sébastien Erckelbout for helping me supervising lab sessions and giving feedback on the teaching content. I would also like to thank the reviewers for their valuable feedback.

References

[AB15] ACKERMANN P., BACH T.: Redesign of an introductory computer graphics course. In *36th Annual Conference of the European Association for Computer Graphics, Eurographics 2015 - Education Papers, Zurich, Switzerland, May 4-8, 2015* (2015), Teschner M., Bronstein M. M., (Eds.), Eurographics Association, pp. 9–13. doi:10.2312/eged.20151021. 2

[AG16] AMADOR G., GOMES A. J. P.: A video games technologies course: Teaching, learning, and research. In *37th Annual Conference of the European Association for Computer Graphics, Eurographics 2016 - Education Papers, Lisbon, Portugal, May 9-13, 2016* (2016), Santos B. S., Dischler J., (Eds.), Eurographics Association, pp. 45–48. doi:10.2312/eged.20161027. 2

[BO15] BADA S. O., OLUSEGUN S.: Constructivism learning theory: A paradigm for teaching and learning. *Journal of Research & Method in Education 5*, 6 (2015), 66–70. 2

[BSP17] BÜRGISSER B., STEINER D., PAJAROLA R.: brenderer: A flexible basis for a modern computer graphics curriculum. In *38th Annual Conference of the European Association for Computer Graphics, Eurographics 2017 - Education Papers, Lyon, France, April 24-28, 2017* (2017), Bourdin J., Shesh A., (Eds.), Eurographics Association, pp. 27–34. doi:10.2312/eged.20171023. 2

[BWF17] BALREIRA D. G., WALTER M., FELLNER D. W.: What we are teaching in introduction to computer graphics. In *38th Annual Conference of the European Association for Computer Graphics, Eurographics 2017 - Education Papers, Lyon, France, April 24-28, 2017* (2017), Bourdin J., Shesh A., (Eds.), Eurographics Association, pp. 1–7. doi:10.2312/eged.20171019. 2

[CXR18] CHEN M., XU Z., RIPPIN W.: On the Pedagogy of Teaching Introductory Computer Graphics without Rendering APIs. In *EG 2018 - Education Papers* (2018), Post F., Žára J., (Eds.), The Eurographics Association. doi:10.2312/eged.20181007. 2

[DC17] DODGSON N. A., CHALMERS A.: Designing a computer graphics course for first year undergraduates. In *38th Annual Conference of the European Association for Computer Graphics, Eurographics 2017*

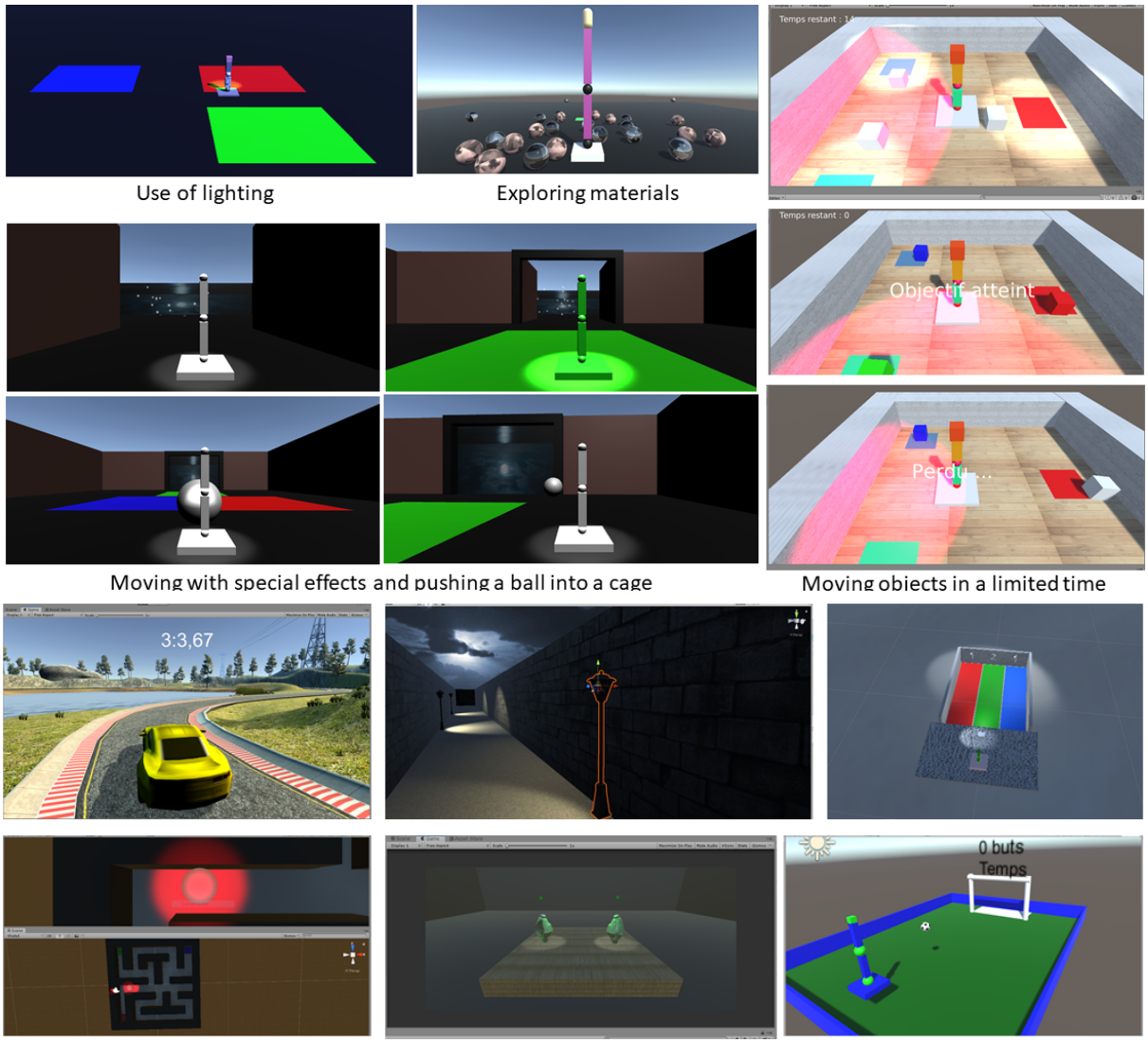


Figure 6: *From top to bottom, example of student results, from simple extensions to the the practical labs to more complex game scenarios.*

- *Education Papers, Lyon, France, April 24-28, 2017* (2017), Bourdin J., Shesh A., (Eds.), Eurographics Association, pp. 9–15. doi:10.2312/eged.20171020. 2

[Dic15] DICKSON P. E.: Using unity to teach game development: When you've never written a game. In *Proceedings of the 2015 ACM Conference on Innovation and Technology in Computer Science Education* (New York, NY, USA, 2015), ITiCSE '15, Association for Computing Machinery, p. 75–80. doi:10.1145/2729094.2742591. 2

[FB09] FELDER R. M., BRENT R.: Active learning: An introduction. *ASQ higher education brief* 2, 4 (2009), 1–5. 2

[Mic] Microsoft teams: Microsoft video conferencing tool. https://www.microsoft.com/en-us/microsoft-teams/group-chat-software. 3

[RMV*21] RODRIGUES R., MATOS T., VALLE DE CARVALHO A., BARBOSA J. G., ASSAF R., NÓBREGA R., COELHO A., DE SOUSA A. A.: Computer graphics teaching challenges: Guidelines for balancing depth, complexity and mentoring in a confinement context. *Graphics and Visual Computing 4* (2021), 200021. doi:https://doi.org/10.1016/j.gvc.2021.200021. 2

[She15] SHESH A.: Teaching graphics to students struggling in math: An experience. In *36th Annual Conference of the European Association for Computer Graphics, Eurographics 2015 - Education Papers, Zurich, Switzerland, May 4-8, 2015* (2015), Teschner M., Bronstein M. M., (Eds.), Eurographics Association, pp. 23–29. doi:10.2312/eged.20151023. 2

[Sim93] SIMONS P.: Constructive learning: The role of the learner. In *Designing environments for constructive learning*. Springer, 1993, pp. 291–313. 2

[SSC02] SLATER M., STEED A., CHRYSANTHOU Y.: *Computer Graphics and Virtual Environments: From Realism to Real-time*. Addison-Wesley, 2002. 3

[Uni] Unity real-time development platform. https://unity.com/. 1, 2, 3