PyXYZ: An Educational 3D Wireframe Engine in Python

Diogo de Andrade
Lusófona University
School of Communication, Arts and Information Tech.
Lisboa, Portugal
diogo.andrade@ulusofona.pt

Nuno Fachada
Lusófona University
COPELABS
Lisboa, Portugal
nuno.fachada@ulusofona.pt

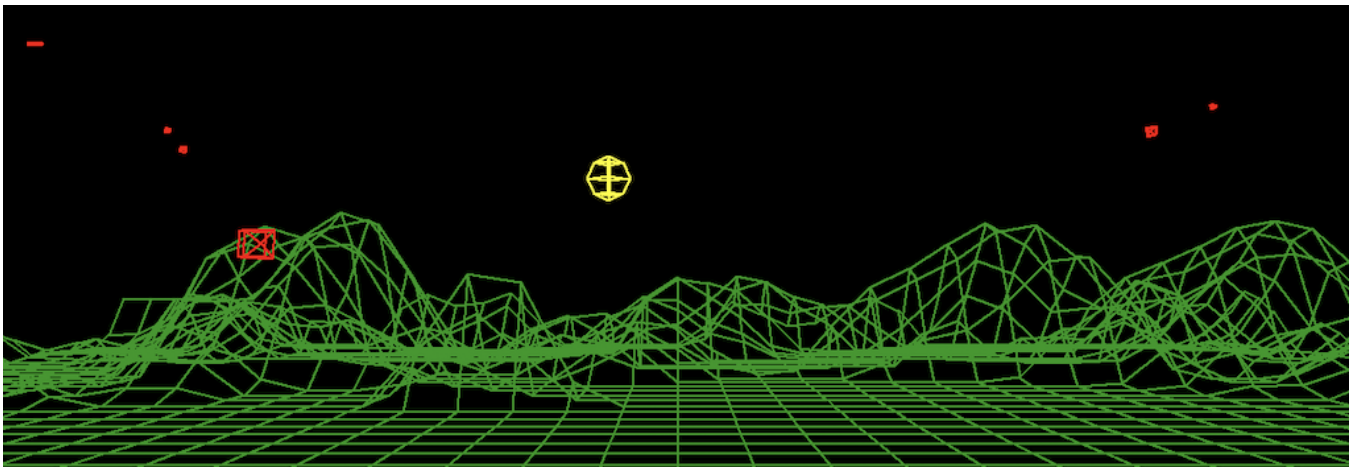

Figure 1: Sample game application for PyXYZ, where the player has to destroy missiles using the mouse.

ABSTRACT

In this paper we introduce PyXYZ, a 3D wireframe software rendering framework for educational purposes. The main goal of this framework is to provide a simple-to-understand tool that students can use to build a more sophisticated engine, while learning mathematics and acquiring a deeper knowledge of the complexity of a modern 3D engine. PyXYZ can be used as a teaching aid in course work and/or as a template for multi-goal project assignments, allowing students with diverse capabilities and interests to have different levels of commitment. The engine has been used with positive results in a mathematics course unit of a computer games BA and can be easily adapted to various teaching scenarios.

CCS CONCEPTS

• **Applied computing** → **Computer games**; **Education**; • **Mathematics of computing**; • **Software and its engineering** → *General programming languages*;

KEYWORDS

software rendering, computer games, 3D, undergraduate degree, Python, Pygame

ITiCSE 2021, June 26-July 1, 2021, Virtual Event, Germany.

ACM ISBN 978-1-4503-8214-4/21/06.
https://doi.org/10.1145/3430665.3456345

ACM Reference Format:
Diogo de Andrade and Nuno Fachada. 2021. PyXYZ: An Educational 3D Wireframe Engine in Python. In *Proceedings of the 2021 ACM Conference on Innovation and Technology in Computer Science Education (ITiCSE '21), June 26-July 1, 2021, Virtual Event, Germany.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3430665.3456345

1 INTRODUCTION

Teaching mathematics to students of different backgrounds is a difficult task [3]. It is also a requirement at Lusófona University's Bachelor in Videogames, a three year multidisciplinary Bachelor of Arts (BA), open to students with various high school academic backgrounds, such as arts, sciences, or humanities [10]. Consequently, in the past, a significant part of the students has struggled with understanding mathematics, as well as its relevance for developing computer games and for their own future careers.

In this paper we introduce PyXYZ (read *pixies*, pronounced ˈpɪksiz), a simple 3D wireframe engine for education, entirely programmed in Python. The engine is focused on the transformation part of the rendering pipeline and has simplicity and increasing student engagement with mathematics as its main goals. It can be used in two ways:

(1) As a teaching aid in course work, for example to demonstrate matrix multiplication.
(2) As an evaluation tool, in which students are required to add functionality.

The engine uses a software-based renderer instead of modern GPU rendering. Using APIs such as OpenGL would add a layer of abstraction and hide the transformation module of the rendering

pipeline. Considering more advanced topics (such as shaders) require solid knowledge of these components, the decision was made to avoid hardware APIs and, by extension, existing game engines, as most are focused on functionality and performance, leading to optimized code that is harder to read and modify.

The paper is organized as follows. In Section 2, we review related work concerning the use of game engines in education. In Section 3, we present the PyXYZ engine, discussing its architecture, dependencies, basic usage, limitations, included sample applications and availability. A field trial in which PyXYZ was used for the final project of a math for games course unit is presented in Section 4. A discussion of the design decisions and how the engine can be adapted for different contexts takes place in Section 5. The paper closes with the Conclusions in Section 6.

2 BACKGROUND

There is extensive work on using existing and/or custom-made game engines for teaching game development fundamentals such as the math of rendering. Parberry et al. [21] discuss the advantages and disadvantages of using an existing engine or implementing your own. Later, Parberry [20] presented the custom SAGE engine, and its use on a number of game development subjects, namely a math and physics for games course. Other custom-designed frameworks include Fabula [4] and wxgame2 [13], but most of these are focused on more general game prototyping and development, and not necessarily on the math and rendering aspects targeted by this work. The engine developed by Aycock et al. [2] targets CS1 students and shares some of the same goals as PyXYZ, but it focuses on 2D and fake 3D games, while we are mostly interested in the actual 3D transformations.

Alice [7] has an educational goal and a target audience composed of non-science/engineering undergraduates, and uses Python to manipulate a 3D environment. However, its focus is on content production while avoiding mathematics and programming, laying down the groundwork for future use of commercial game engines. This is also the case of platforms such as AgentCubes [15]. Consequently, both tools have opposite goals to PyXYZ.

Depending on course requirements, an existing game engine can often be more productive. Several authors have proposed a number of guidelines to aid in selecting the engine, namely its cost (or the absence thereof), quality of documentation, flexibility and extensibility, cross-platform support, learning curve, usage in industry and runtime stability [9, 25].

The Unity game engine meets many of the aforementioned guidelines. Indeed, it has been successfully used in many education scenarios, often proving more advantageous and productive than other engines [9] or custom approaches [14]. For example, it has been used in secondary education [6] and for teaching shader programming [14], and is in fact the game engine of choice for most Lusófona's Videogames degree course units [10]. However, like most of the available engines, Unity is designed for efficiency and, therefore, uses the GPU through a graphics API such as OpenGL, DirectX or Vulkan, hiding many of the implementation details we want students to explore.

Since the first semester of Lusófona's Videogames degree is focused on the Python programming language [10], a number of Python-based engines were also considered. These range from simple OpenGL bindings like PyOpenGL [11], to full engines like Panda3D [23], Py3D [12] and Ursina [1]. Unfortunately, most of these are actually written in C/C++ and have Python bindings, making it harder for students to tinker with the transformation pipeline, which is the intended goal of PyXYZ. In these engines the pipeline is hidden behind the API, and manipulating it would require a deeper understanding of GPU and shader programming.

At the time of writing, and to the best of our knowledge, there was no generally available and well-documented Python 3D engine that shares the design philosophy and educational goals of PyXYZ, particularly its focus on teaching the transformation pipeline.

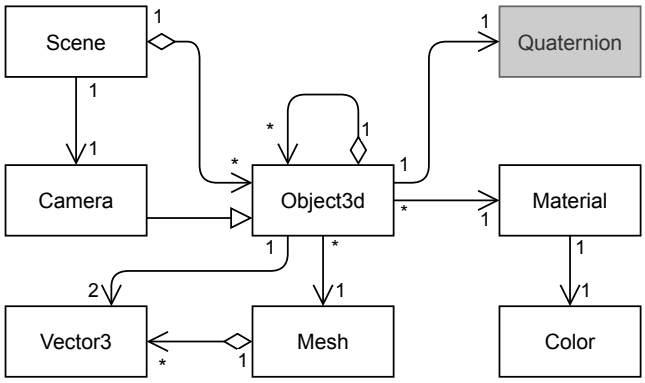

Figure 2: PyXYZ UML diagram. The `Quaternion` class is presented in grey since it is provided by a third-party library.

3 PYXYZ

3.1 Architecture

PyXYZ is an object-oriented engine. Its main design focus is on simplicity, ease of learning and extension. It provides very little functionality out-of-the-box, basically allowing for the programmer to visualize a 3D scene using a virtual camera. A scene is composed of 3D objects organized in an optional hierarchical fashion, and each object contains a polygonal mesh and a material that controls how the mesh is rendered. Figure 2 shows the UML diagram representing the general architecture of the engine.

PyXYZ provides two elementary helper classes: `Color`, which describes a color with separate red, green, blue and alpha channels; and `Vector3`, a straightforward 3D vector implementation.

The core of the engine comprises the `Scene`, `Object3d`, `Camera` and `Mesh` classes, which handle the rendering itself. An `Object3d` has the position, rotation and scaling (PRS properties), all of which are in local space. While position and scaling are represented by a `Vector3`, rotation is handled by a `Quaternion` instance (the `Quaternion` class is provided by a third-party library, as discussed in the next section). An `Object3d` also stores the reference for a mesh and a material, and contains a list of child `Object3d`, which enables the user to build the hierarchical scene graph. A `Scene` stores the scene graph with any number of `Object3d` instances on the root level. It also contains a camera, used for the rendering.

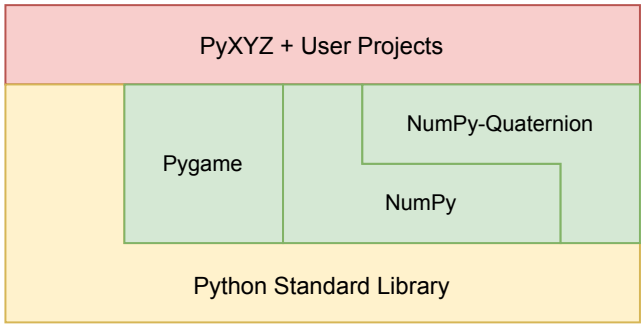

Figure 3: PyXYZ and user project dependencies. Users develop their projects in the top-level (light red block), possibly manipulating PyXYZ source code and being exposed to the Pygame, NumPy and NumPy-Quaternion libraries (light green), as well as to the Python Standard Library (light yellow).

The `Camera` class is derived from `Object3d`, so that it can be treated in the same way, and even parented to other objects, or vice-versa. It also provides a simple function to convert from screen space coordinates to a ray origin/position. Note that the camera can be parented to other objects, but the rendering itself uses the local PRS of the camera, and not the world PRS. This is one of the possible exercises given to the students.

The `Mesh` contains a list of polygons, with each polygon being a list of vertex positions in local space. There are no indexing primitives as simplicity is the main driver of the engine. The `Material` class stores rendering properties such as line color and width. A single material can be used by multiple meshes for rendering.

3.2 Dependencies

PyXYZ uses Pygame [17] as a visualization engine. It also uses NumPy [19] and NumPy-Quaternion [5] for matrix and quaternion management, respectively. All of these libraries are visible at the user level, so they can be used directly for developing projects and extensibility purposes, as shown in Figure 3.

Pygame is used for the actual rendering. It was selected for its simplicity and support for polygon rendering. It also performs input handling, allowing students to build interactive systems – a requirement in several projects.

NumPy and NumPy-Quaternion are used for matrix multiplication and quaternion operations, respectively, and were chosen for performance reasons. Nonetheless, NumPy is not used for user-facing vector handling since the syntax is a bit complex and it might overwhelm the students. The use of NumPy is only required when delving deeper into the engine or when implementing advanced features such as clipping or lighting. There is, however, some exposure to NumPy-Quaternion, since this is the way to handle rotations in PyXYZ.

The chosen libraries are all cross-platform and fully open source, allowing PyXYZ to be used on different computational systems and without proprietary restrictions.

Listing 1: Creating a scene, a camera, and an object that contains a sphere mesh.

```
# Create a scene
scene = Scene("TestScene")
scene.camera = Camera(False, 640, 480)

# Moves the camera back 2 units
scene.camera.position -= Vector3(0, 0, 2)

# Create a sphere and place it in a scene, at
# position (0,0,0)
obj1 = Object3d("TestObject")
obj1.scale = Vector3(1, 1, 1)
obj1.position = Vector3(0, 0, 0)
obj1.mesh = Mesh.create_sphere((1, 1, 1), 12, 12)

# Set the material of the sphere, in this case it is red
obj1.material = Material(color(1, 0, 0, 1),
                         "TestMaterial1")
scene.add_object(obj1)
```

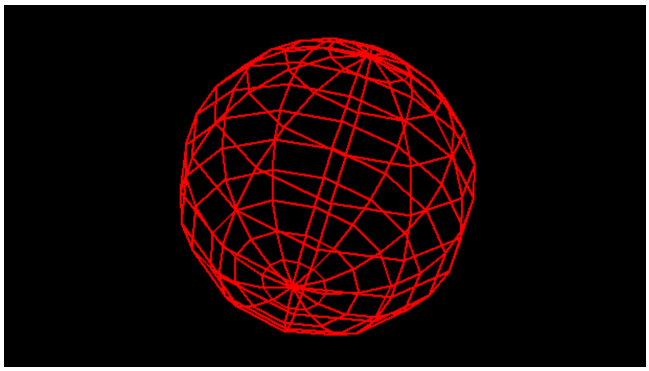

Figure 4: Sample sphere application for PyXYZ, based on the code shown in Listing 1.

3.3 Basic usage and rendering process

From an engine perspective, the programmer just needs to initialize the Pygame context, using code similar to:

```
pygame.init()
screen = pygame.display.set_mode((640, 480))
```

Then, the programmer can create a scene and add objects to it, as shown in Listing 1. This example leads to a rendering like in Figure 4, which can be drawn on screen with the following instruction:

```
scene.render(screen)
```

For simplicity, instead of following the example of modern rendering engines, this does not trigger a gather process and building of a command queue for rendering, opting instead for a simple hierarchical traversal of the scene graph and requesting every object to render itself, computing the clip matrix at every graph node. The meshes that are defined at every level are then rendered using this clip matrix.

All the transformation and rendering are done at a software level. Again, for simplicity's sake, this is a two-step process:

(1) Transform mesh vertices.

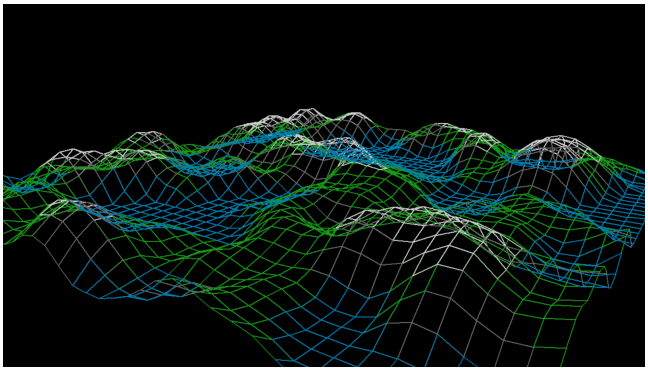

Figure 5: Sample terrain application for PyXYZ, generated with a 2-octave Perlin noise, and with colors based on the height and slope of the polygon.

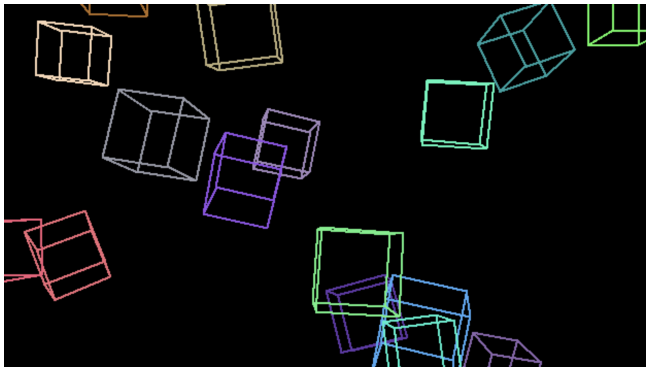

Figure 6: *Cubefall* sample application for PyXYZ. Cubes are spawned over time and fall down, being destroyed when they leave the screen.

(2) Render the transformed mesh vertices.

No indexing is done at the mesh level, although it would be easy to add. This might be given as a student exercise, which is the main reason why the transform and render steps are separated.

Note that, since the engine does not use a command queue approach, there is no sorting step. For a wireframe engine, this is irrelevant, but as we ask students to add solid-polygon rendering, they will have to change this part of the pipeline to add sorting. Primitive clipping is also not implemented, so there may be artifacts when objects are rendered behind the camera. Implementing frustum culling and clipping may also be proposed as an exercise for more advanced students.

3.4 Sample applications

Even with the limitations imposed by the desired simplicity of the engine, some interesting graphical applications can be built. A number of sample applications are available, serving not only to demonstrate PyXYZ's capabilities but also to exemplify how to use the engine. For example, the sphere sample application (Listing 1, Figure 4) shows how to use PyXYZ's most basic capabilities.

A more interesting sample application is shown in Figure 5. This application generates a terrain based on a 2-octave Perlin noise, colors the generated polygons based on height and slope, and rotates the terrain in front of the camera.

Another sample is the *cubefall* example, shown in Figure 6. Here, cubes are generated on the upper part of the camera viewpoint and drop down with gravity. This example shows the students how to manage object lifecycle and animation, instead of following the standard pattern of creating a scene and visualizing it.

The most complex sample is a template for a game where the player can shoot some missiles using the mouse, as shown in Figure 1. The project demonstrates user input and a more complex application loop. It also shows functionality like creating a mesh from different parts, converting a mouse position into a ray, sphere/sphere collision detection, target tracking and a rudimentary screen flash effect. The game is not complete, since it does not have win/lose conditions, nonetheless opening a number of possibilities for student projects.

More sample applications can potentially be made available, further exemplifying the capabilities of and how to use PyXYZ. However, new examples should stay relatively simple, so that students may develop new functionalities by themselves. As an example, applications in which the camera goes inside the scene being rendered, and thus requiring frustum culling/clipping, should not be included, allowing for such functionality to be implemented by students as an exercise or as a project requirement.

3.5 Limitations

A considerable limitation is the fact that pure, interpreted Python is slow, so the engine will never be able to have good performance. The terrain sample application was benchmarked to understand the severity of this limitation, measuring the frame time (spanning from start of rendering until frame is displayed on screen, so it might include vertical sync), the transformation time (transforming the mesh vertex from local space to clip space) and render time (time to actually render the wireframe polygons). All these benchmarks were performed on a 2019 Apple MacBook Pro with an 8-core Intel i9 processor, running at 2.3 GHz, with 32 GB of DDR4 RAM. PyXYZ and Pygame make no use of multithreading, so only a single core is used.

As it is clear from Figure 7, the frame time scales linearly with the vertex count. The bulk of the frame time is spent on the transformation step. This is unsurprising, given that the rendering is done by Pygame and is implemented natively (Pygame is essentially a wrapper to a C library), while transformation is done using a pure Python loop to iterate the vertices and multiply them by the clip matrix. Although this vector/matrix multiplication is implemented through NumPy (and thus optimized), the loop itself is a bottleneck in an interpreted language such as Python, and a prime candidate for vectorization. Although it would not be difficult to do so using NumPy's capabilities, the code would become harder to read and modify, which is in contrast with the primary goal of this engine, so the decision was made to prioritize clean, readable code over optimization.

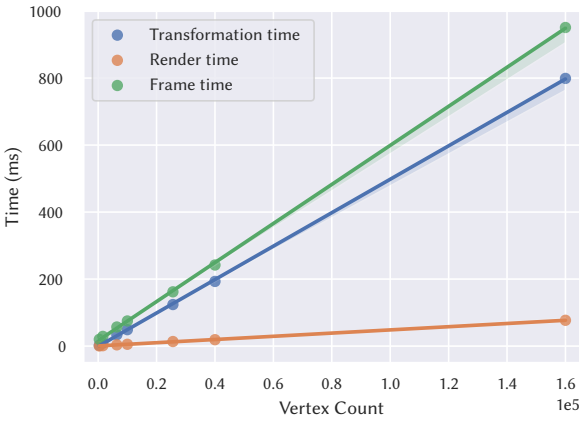

Figure 7: Performance benchmark of PyXYZ with increasing vertex counts.

It might be an option for more advanced students to challenge them with implementing vectorization of the mesh rendering, especially considering that when lighting is implemented, there is additional transformation that must be done to account for world space normals.

Nonetheless, even with these performance limitations, the engine is usable for simple scenarios and games. We do not believe simplicity and ease of use should be sacrificed for better performance when the goal is "simply" to motivate students to learn math.

3.6 Availability

PyXYZ is made available as free and open source software under the MIT license at https://github.com/VideojogosLusofona/PyXYZ. Full source code of the examples presented in Section 3.4, including instructions on how to run them, is available at https://github.com/VideojogosLusofona/PyXYZ-Samples. The project is fully documented and the documentation is available at https://videojogoslusofona.github.io/PyXYZ/.

4 FIELD TRIAL

The curriculum of two course units at Lusófona University's Bachelor in Videogames, Math and Physics for Games I (MPG1) and Programming Fundamentals (PF), was collaboratively designed so that the courses could work in tandem [10]. In MPG1 we teach basic topics such as trigonometry and matrix mathematics. In PF, students learn how to program using the Python programming language, focusing mainly on algorithms and basic data structures. The dual curriculum design advocates that the mathematical concepts exposed in MPG1 drive the examples and exercises in PF, thus allowing students to experiment first-hand with often abstract math topics. Complementing this approach, the overall evaluation in MPG1 includes a project that combines knowledge in both mathematics and programming. The need for a layered and multi-goal assignment, accessible to students with diverse capabilities and interests, was the main motivation for the work presented here.

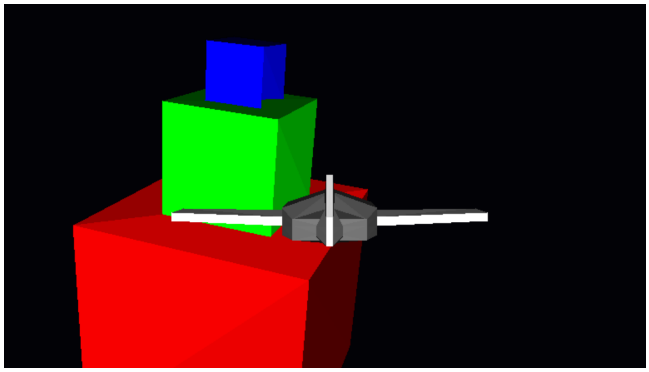

Figure 8: Screenshot of a student's final project.

PyXYZ was used in the 2019/2020 MPG1 final assignment, with a set of tasks to be completed by the students—some of which are presented in Table 1, together with their level of difficulty. Evaluation was tiered and linked to these tasks. A total of 28 students were evaluated. With respect to the practical evaluation component, there were 16 assignments handed in (corresponding to 22 students), 15 of which were rated with a passing grade. Grading was done on a 0 to 20 scale. Most grades were around the 10 to 13 mark (mean 11.8, median 12), corresponding to students that completed the most basic part of the assignment with one extra assignment goal, at most. While the assignment could be undertaken by groups of up to three students, most of the higher grades (>15) were for individual submissions. Students were later asked to rate their motivation [22, 24] with the project, in a scale of 1 to 5. The gathered data, spanning students from the last three academic years, in which only the last year (2019/2020) used PyXYZ in the project, showed that students were more motivated, with the mean motivation going up, from 2.55 to 3.25 [8].

The use of a practical example that can be taken apart and extended is a very helpful tool for educational purposes, especially in the context of a broad-spectrum videogames degree, where the interests and motivations are different from student to student. Some students are driven by a short term reward cycle, while others thrive on understanding complex topics and by tinkering. As an example, one of the students rebuilt a significant part of PyXYZ to make it more similar to the Unity game engine [16], in which he was already proficient, namely by reimplementing game objects using the Component pattern [18]. A screenshot of this project is shown in Figure 8.

5 DISCUSSION

Most of the mathematical concepts in the curriculum are mainly used in the context of 3D graphics and physics. For that reason, the possibility of using 2D engines for this part of the curriculum was discarded. Nonetheless, these are used in course units geared towards game design and general game development.

Python was chosen instead of a lower-level language since it is easy to learn, and we want students to focus on higher-level concepts instead of being worried with low-level memory management

Table 1: Some of the more relevant tasks for the final assignment of the 2019/2020 Maths and Physics for Games I course unit, together with their level of difficulty, ranked from 1 to 5.

Task	Difficulty
Create a new geometric figure (pyramid, star, cylinder)	1
Create a viewer for the figure (user-controlled object rotation and translation)	1
Create a FPS controller (mouse and keyboard control)	2
Implement backface culling	3
Implement filled geometry and object sorting	4
Implement per-vertex point lighting with a Lambertian model	5

and optimization. These issues are addressed in another course unit [10].

As already stated, the NumPy and NumPy-Quaternion libraries were selected for performance reasons. However, an alternative approach would be to remove these dependencies, replacing them with pure Python code. These libraries add a layer of complexity and hide details that might be useful to expose students to. The loss of performance might be offset by the increased readability, and especially by allowing students to observe the internals of matrix multiplication and quaternion math working in a real context – something important at this stage of their education. Alternatively, some sort of wrapper might be added to get the best of two worlds.

Another avenue being explored is hiding all of Pygame behind a wrapper, so that students do not need to be aware of it while using PyXYZ. Although our BA students learn Pygame in the PF course unit, this change would widen PyXYZ's usefulness in scenarios where this library is not part of the curriculum. Thus, taking this idea further, it might be worth building a complete layer that takes care of the application setup and loop that abstracts the underlying visualization and input library.

Custom polygon rendering, as an alternative to Pygame, would also be a useful feature. This would turn PyXYZ into an excellent tool for introducing students to the rasterization pipeline, instead of focusing only on the transformation part of the rendering. Although this idea has allure to it, there are performance concerns to consider, so some experiments are required to determine if this is feasible.

Every curriculum has its own requirements and needs. The alternative approaches discussed in this section are not meant as definitive improvements to PyXYZ, but as paths which can be followed to adapt PyXYZ to other teaching scenarios. In this regard, PyXYZ can be considered not only a game engine template for students to modify, create simple 3D games and learn math, but also as a teaching tool template for educators to adapt to their needs.

6 CONCLUSIONS

In this paper we presented PyXYZ, a Python-based 3D wireframe engine developed with the goal of supporting the teaching of mathematics and the transformation part of a rendering pipeline to game development students from a wide variety of backgrounds.

PyXYZ can be a powerful tool for motivating students, as discussed in Section 4. It makes it relatively simple to achieve something interesting by modifying and playing around with the sample applications, while stimulating students that are more interested in the engineering part of the game development process to create more complex applications. Although most students will end up using commercial engines and for the most part not required to know the intricacies of modern rendering, there are some tasks that demand a knowledge of the rendering pipeline, like shader programming. The insights gained while working on projects with PyXYZ will help them understand the importance of mathematics in game development, as well as the potential and limitations of 3D game engines.

Finally, educators can adapt PyXYZ to different teaching contexts, for example improving its performance by further integrating optimized numerical libraries such as NumPy and NumPy-Quaternion; or, conversely, making the code clearer by using the Python Standard Library for all vector and matrix operations. Even the Pygame dependency can be removed, making PyXYZ a tool for studying the rasterization pipeline via custom polygon rendering. In short, PyXYZ can be viewed as a starting point and framework which educators can adjust to their curricula.

ACKNOWLEDGMENTS

The authors would like to thank André Fachada for proof-reading the text. The authors would also like to thank the anonymous referees for their valuable comments and helpful suggestions. This work is supported by Fundação para a Ciência e a Tecnologia under Grant No.: UIDB/04111/2020 (COPELABS).

REFERENCES

[1] Petter Amland. 2017. Ursina. GitHub. https://github.com/pokepetter/ursina Accessed: 2021-01-12.
[2] John Aycock, Etienne Pitout, and Sarah Storteboom. 2015. A Game Engine in Pure Python for CS1: Design, Experience, and Limits. In *Proceedings of the 2015 ACM Conference on Innovation and Technology in Computer Science Education* (Vilnius, Lithuania) *(ITiCSE '15)*. ACM, New York, NY, USA, 93–98. https://doi.org/10.1145/2729094.2742590
[3] Sarah E. Bamforth, Carol L. Robinson, Tony Croft, and Adam Crawford. 2007. Retention and progression of engineering students with diverse mathematical backgrounds. *Teaching Mathematics and Its Applications: International Journal of the IMA* 26, 4 (2007), 156–166. https://doi.org/10.1093/teamat/hrm004
[4] Florian Berger and Wolfgang Müller. 2012. Towards an open source game engine for teaching and research. In *Transactions on Edutainment VIII*. Springer, 69–76.
[5] Mike Boyle. 2018. The quaternion package: Add support for quaternions to python and numpy (Version v2.0). Zenodo. https://doi.org/10.5281/zenodo.1220425
[6] Oswald Comber, Renate Motschnig, Hubert Mayer, and David Haselberger. 2019. Engaging Students in Computer Science Education through Game Development with Unity. In *2019 IEEE Global Engineering Education Conference (EDUCON)*. IEEE, 199–205. https://doi.org/10.1109/EDUCON.2019.8725135
[7] Matthew Conway, Steve Audia, Tommy Burnette, Dennis Cosgrove, and Kevin Christiansen. 2000. Alice: lessons learned from building a 3D system for novices. In *Proceedings of the SIGCHI conference on Human Factors in Computing Systems* (The Hague, The Netherlands) *(CHI '00)*. ACM, New York, NY, USA, 486–493. https://doi.org/10.1145/332040.332481

[8] Diogo de Andrade and Nuno Fachada. 2020. Fun maths for all game development students. In *Proceedings of the 2020 ACM Conference on Innovation and Technology in Computer Science Education* (Trondheim, Norway) *(ITiCSE '20)*. ACM, New York, NY, USA, 529–530. https://doi.org/10.1145/3341525.3393992

[9] Paul E. Dickson, Jeremy E. Block, Gina N. Echevarria, and Kristina C. Keenan. 2017. An Experience-based Comparison of Unity and Unreal for a Stand-alone 3D Game Development Course. In *Proceedings of the 2017 ACM Conference on Innovation and Technology in Computer Science Education* (Bologna, Italy) *(ITiCSE '17)*. ACM, New York, NY, USA, 70–75. https://doi.org/10.1145/3059009.3059013

[10] Nuno Fachada and Nélio Códices. 2020. Top-down design of a CS curriculum for a computer games BA. In *Proceedings of the 2020 ACM Conference on Innovation and Technology in Computer Science Education* (Trondheim, Norway) *(ITiCSE '20)*. ACM, New York, NY, USA, 300–306. https://doi.org/10.1145/3341525.3387378

[11] Mike C. Fletcher. 2000. PyOpenGL. Sourceforge. http://pyopengl.sourceforge.net/ Accessed: 2021-01-12.

[12] Adam Griffiths. 2012. Py3D. GitHub. https://github.com/adamlwgriffiths/Py3D Accessed: 2021-01-12.

[13] Junyoung Heo and Seukwon Kang. 2015. Simple shooting game engine in Python. *International Journal of Computational Vision and Robotics* 5, 2 (2015), 130–137.

[14] Mitja Hmeljak. 2020. Developing a Computer Graphics Course with a Game Development Engine. In *Proceedings of the 2020 ACM Conference on Innovation and Technology in Computer Science Education* (Trondheim, Norway) *(ITiCSE '20)*. ACM, New York, NY, USA, 75–81. https://doi.org/10.1145/3341525.3387428

[15] Andri Ioannidou, Alexander Repenning, and David C. Webb. 2009. AgentCubes: Incremental 3D end-user development. *Journal of Visual Languages & Computing* 20, 4 (2009), 236–251. https://doi.org/10.1016/j.jvlc.2009.04.001

[16] Roberto Junior. 2020. Projecto de IMFJ1. GitHub. https://github.com/robertojrdev/imfj1_2019_projecto Accessed: 2021-01-12.

[17] Will McGugan. 2007. *Beginning game development with Python and Pygame: from novice to professional*. Apress.

[18] Robert Nystrom. 2014. *Game programming patterns*. Genever Benning. https://gameprogrammingpatterns.com/

[19] Travis E. Oliphant. 2006. *A guide to NumPy*. Vol. 1. Trelgol Publishing USA.

[20] Ian Parberry. 2011. Challenges and opportunities in the design of game programming classes for a traditional computer science curriculum. *Journal of Game Design and Development Education* 1 (2011), 4–17. Issue 1.

[21] Ian Parberry, Timothy Roden, and Max B. Kazemzadeh. 2005. Experience with an Industry-driven Capstone Course on Game Programming: Extended Abstract. In *Proceedings of the 36th SIGCSE Technical Symposium on Computer Science Education* (St. Louis, Missouri, USA) *(SIGCSE '05)*. ACM, New York, NY, USA, 91–95. https://doi.org/10.1145/1047344.1047387

[22] Paul Pintrich. 2003. A motivational science perspective on the role of student motivation in learning and teaching contexts. *Journal of Educational Psychology* 95, 4 (2003), 667–686. https://doi.org/10.1037/0022-0663.95.4.667

[23] David Rose et al. 2000. Panda3D. Carnegie Mellon University. https://panda3d.org Accessed: 2021-01-12.

[24] Dale Schunk, Judith Meece, and Paul Pintrich. 2013. *Motivation in education theory, research, and applications* (fourth ed.). Pearson Education Limited.

[25] Bian Wu and Alf Inge Wang. 2012. A Guideline for Game Development-based Learning: A Literature Review. *International Journal of Computer Games Technology* 2012, Article 103710 (Dec. 2012). https://doi.org/10.1155/2012/103710