This manuscript has been accepted for presentation in the 13th IEEE Integrated STEM Education Conference, Johns Hopkins University, Laurel, Maryland, March 11, 2023

Developing Mini VR Game Engines as an Engaging Learning Method for Digital Arts & Sciences

Angelos Barmpoutis, Wenbin Guo, Ines Said
University of Florida, {angelos, wenbin, ines}@digitalworlds.ufl.edu

Abstract – **Digital Arts and Sciences curricula have been known for combining topics of emerging technologies and artistic creativity for the professional preparation of future technical artists and other creative media professionals. One of the key challenges in such an interdisciplinary curriculum is the instruction of complex technical concepts to an audience that lacks prior computer science background. This paper discusses how developing small custom virtual and augmented reality game engines can become an effective and engaging method for teaching various fundamental technical topics from Digital Arts and Sciences curricula. Based on empirical evidence, we demonstrate examples that integrate concepts from geometry, linear algebra, and computer programming to 3D modeling, animation, and procedural art. The paper also introduces an open-source framework for implementing such a curriculum in Quest VR headsets, and we provide examples of small-scale focused exercises and learning activities.**

Index Terms –Computer Education, Digital Arts, Game Engines, Virtual Reality

INTRODUCTION

Since the first use of computing systems in academia, scholars have stressed the importance of integrating computer programming within the Fine Arts and Liberal Arts curricula. In 1962 Perlis argued that computer programming is as essential in a liberal arts course as in science or engineering because it is an educational subject about constructing, analyzing, and describing processes [1]. Buckingham, in 1965, demonstrated how computer programming could be integrated with the Fine Arts by providing the source code and output of a geometric design program [2].

Along the lines of these recommendations, several interdisciplinary programs were formed in the last decades of the twentieth century by extending the traditional fine arts with emerging digital media. The increasing use of computer graphics in motion pictures and the rapid growth of the video game industry led to the further expansion and formalization of these programs in the first decade of the twenty-first century [3]. Digital Arts and Sciences (DAS) programs intersect with graphic design, visual arts, traditional fine arts, computer science, art and design, and others. As a result, there is a natural flow of students between these areas. Subsequently, digital arts and sciences students have a strong background and in-depth exposure to these areas, but their technical foundation differs significantly from those in traditional engineering fields. Therefore, interdisciplinary programs, such as DAS, have the challenge of providing an integrated curriculum covering digital art topics (such as 2D and 3D digital animation, 3D modeling and texturing, digital storytelling, game design, etc.) as well as many technical topics from computer science, such as computer programming, computer graphics, human-computer interaction, and others, to sufficiently prepare future technical artists, special effects artists, and animators, which are few of the highest-paid professions compared to other arts and design occupations [4].

However, acquiring computer science skills, especially technical skills such as programming, can be challenging, and the learning difficulties have been well-known and discussed in the literature [5]. Within the context of digital arts, it has been shown that combining computer programming and art can improve student learning outcomes and effectively teach other technical topics, such as mathematics [6]. Therefore, visual designers and artists can benefit directly by integrating principles from their own discipline with computer programming instruction [7].

Additionally, using games as a learning tool has been very popular, especially in K-12 education, and their benefits have been well studied [8,9,10,11]. Within the CS context, games have been used in two different ways: 1) gamification of curriculum, as a method for motivating and rewarding the students while learning [8,9,14], and 2) game development, as a hands-on experience for learning technical topics [10,11,15], which is the focus of this paper.

Games can be easily developed even by beginners with the use of appropriate tools. Scratch is a popular example of a block coding environment developed at MIT for early exposure to computer programming, which provides several tutorials for making simple games such as chase games, clicker games, pong games, and adventure games [10]. According to Casey Rea and Ben Fry [7], undergraduate students at UCLA learned new programming skills by producing video games or kinetic artworks using Processing, a simple Java-based coding language. A similar environment for beginners is the Python PyGame framework [11].

Moreover, immersive gaming systems such as virtual reality (VR) headsets combine all the aforementioned benefits by integrating 3D arts and games in an immersive and engaging modality and have been used in various educational applications beyond DAS [12,17]. These

immersive interfaces have been shown to increase the level of engagement by transforming the learning content into hands-on experiences [12].

Game development software, such as Unreal and Unity 3D, offer VR game engines and collections of tools and automations that simplify the game development process and thus have become an important tool of instruction in DAS curricula [15]. Although these tools can be used professionally to produce commercial-standard games, their automated pipeline can also be helpful to beginners, as it is possible to produce a playable game with minimal effort [16]. Students can choose from a variety of available 3D models, prepackaged shaders, preprogrammed colliders, game components, and other plugins without the burden of designing them or coding them from scratch and thus avoid blank page trauma.

However, technical artists and other creative media professionals, more often than not, are expected to be able to design and animate their own models, code custom shaders, program colliders, and other components beyond those available off-the-shelf. So, the main question is, where do we draw a line between consuming existing designs and solutions and creating new ones in an academic curriculum? For example, should a post-beginners level student stop using premade shaders and start exploring the inner technical structures of a shader to learn how to create new shaders in GLSL? And what are the tools that we need to provide to the students to be able to explore the technical details that are usually hidden in premade game engines?

This paper argues that the process of making or modifying a custom-made small-scale game engine can be of significant educational value because it lets students explore and understand several necessary technical concepts that are typically left unexplored when using premade game engines. The contributions of this paper are three-fold: 1) We present several examples of focused learning activities that demonstrate how specific technical concepts related to game development can be taught in practice. 2) We introduce an open-source framework for implementing such a curriculum in Quest VR headsets. Finally, 3) We present our findings from a limited testing of this framework in an undergraduate DAS course.

Making a Basic Game Engine as a Learning Method

Creating a VR game engine is a complex task that requires the development of several components, including geometric structures (models and scene hierarchy, transformations, etc.), rendering processes (shaders, shadow generators, etc.), file format parsers (3D objects, textures, etc.), animation processes (physics simulations, particle effects, etc.), hardware event listeners (controllers, headset, triggers, etc.), and many others. Hence, the use of an existing game engine is the recommended choice, especially for quick prototyping and production. However, when students rely on an existing body of work, some foundational elements may be left unexplored within an educational context. Therefore, it is essential to be given the opportunity to study these foundational elements and understand how they function.

One way to do this is by exploring the source code of a particular VR game engine component (for example, a collider function or a fragment shader) and experimenting with it by modifying it and observing the resulting behavior. However, in many cases, the source code may not be available, or it is so advanced and complex that it is hard to understand or cannot be modified and recompiled easily.

Therefore, having a readily available and easy-to-modify small-scale game engine can be very educational, especially if it is used as part of targeted learning activities. Furthermore, the following characteristics could improve its usability and ease of use within educational settings:

- *Self-contained and ready to use.* The framework must contain all necessary resources to minimize the setup effort.
- *Open-source*. The source code of the game engine components must be available and easy to compile.
- *Minimum functionality.* The framework must contain a minimal game engine implementation to keep each component simple and easy to modify.
- *Well-structured and -documented.* Each component must be structured in an intuitive object-oriented manner and be well-documented.
- *Used within an Integrated Development Environment* to be easy to navigate and extend.
- *Contain examples and templates.* The game engine must come with several examples to facilitate learning by example and minimize blank page trauma.
- *Compatible with VR hardware.* The framework must directly compile and run in modern VR equipment to facilitate fast iterative modify-and-run learning cycles.
- *Non-VR alternative.* To accommodate a broader audience that may not have access to VR hardware, a low-cost non-VR solution must also be available in the same framework.

The following section provides examples of focused learning activities that can be done with the proposed framework.

Examples of Learning Activities

I. 3D coordinate systems and transformations

Linear Algebra, specifically vertex transformations with 4x4 matrices, plays a major role in computer graphics. Such a purely algebraic topic can be better understood by extending the classical Logo turtle graphics exercises [13] to 3 dimensions. Turtle graphics has been a popular API that allows programmers to compose a 2D image by instructing the "turtle" to move on the plane and draw its trace by a sequence of commands such as move forward, turn left, and turn right. It has been shown [7,13] that such exercises can be helpful in teaching procedural logic, coordinate systems, and other topics related to geometry and procedural arts.

These exercises can be extended to 3 dimensions by combining 3D transformation commands (translate, rotate, scale, etc.) with primitive geometric shape commands

(cuboid, sphere, cylinder, pyramid, prism, trapezoid, etc.). In this framework, a 3D shape can be composed by a sequence of commands that move the coordinate system (a.k.a. "the turtle") to the proper position and orientation and place there a primitive shape with the desired color and size, and then move to a different location and put a different shape, etc. Models composed with this process are shown in Fig. 1. This exercise can be helpful in learning 3D transformations and understanding their results, especially when used in sequence. Mastery of coordinate transformation functions is essential for technical artists, especially for creating animations and implementing interactions.

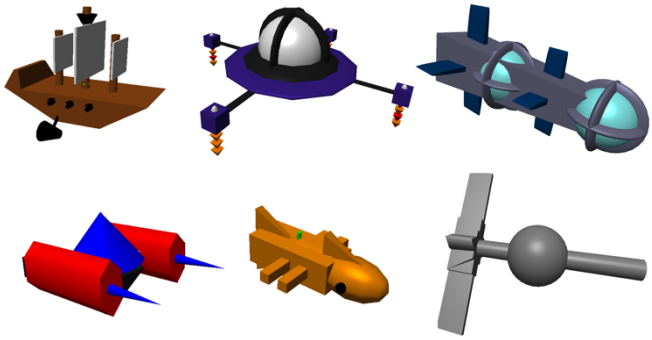
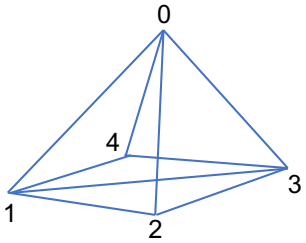


FIGURE I
EXAMPLES OF 3D SHAPES COMPOSED BY STUDENTS USING PRIMITIVE SHAPES AND 3D TRANSFORMATIONS.

II. 3D model encoding

Mastering 3D modeling is an essential skill for digital artists that is typically taught and practiced using a 3D modeling software tool such as Blender (blender.org) and Maya (autodesk.com). In addition to the creative process of designing a tridimensional geometric shape, there are many technical tasks, such as texture mapping, normal mapping, material design, and others that combine concepts from mathematics (such as geometry), physics (such as light reflectance), and technology (computer graphics/shaders) with artistic creativity. The produced 3D model is often saved in a text-based file format, such as OBJ (Wavefront), DAE (COLLADA), or the ASCII versions of PLY (Stanford Triangle Format) and STL (Stereolithography).



5 Vertices	6 Triangles
0: 0, 2, 0	0, 1, 2
1: -1, 0, -1	0, 2, 3
2: -1 ,0, 1	0, 3, 4
3: 1, 0, 1	0, 4, 1
4: 1, 0, -1	1, 4, 3
	1, 3, 2

FIGURE II
DESIGN EXERCISE OF A 3D PYRAMID BY IDENTIFYING THE X,Y,Z VERTEX COORDINATES AND THE VERTEX INDICES IN EACH TRIANGLE.

Browsing the contents of such files with a plain text editor is a practical exercise that exposes to the artists the technical side of their designs, i.e., the coordinates of each vertex, the indices of the 3 vertices in each triangle in the mesh, pixel coordinates in texture mappings, and possibly other material parameters. Generating these parameters manually for a simple geometrical shape, such as a pyramid (Fig. 2), can help students bridge the gap between 3D art and the underlying technology (in this example, 3D data encoding), which is an essential step for understanding rendering using shader languages, such as GLSL.

III. User Interaction

Modern gaming systems offer natural user interfaces that track the user's body (head, hands, etc.) in real-time. They may also have controller devices that provide numerous user input forms, such as toggle buttons, squeeze buttons, and joysticks. Each of these input points generates events that game developers can handle to implement interesting user interactions. Providing an easy-to-use event-based API facilitates fast prototyping by creating an abstraction layer on top of lower-level commands, such as OpenXR library commands.

Several hands-on exercises can be designed for teaching various concepts from Digital Arts and Sciences curricula using this experiential learning setup. One example is the topic of collision detection between 3D objects, an essential task related to human-computer interaction in virtual reality. Developing a collider involves concepts related to geometry (convex solids), resource optimization (mesh simplification), and computer programming (collider implementation). Implementing spherical colliders between a hand-held controller and various virtual objects is an easy exercise that covers several of the aforementioned topics. Additionally, handling button and joystick events can prompt the students to investigate innovative forms of user interaction while working on small-scale focused assignments, such as drawing in 3D by tracing the hand-held controller's location, selecting color and material properties, etc.

OPEN-SOURCE IMPLEMENTATION

The proposed framework has been implemented in Android Studio using Oculus OpenXR Mobile SDK (46.0) and is available at: https://github.com/digitalworlds/JavaForQuest. The project is titled Java For Quest (J4Q), and it demonstrates how to create AR/VR applications for Meta Quest headsets using Java in Android Studio. It can be used as a platform to learn and teach the development of a basic game engine from scratch, as well as related topics, such as 3D geometry, shaders using GLSL, event-driven human-computer interaction, vibration feedback, etc.

The repository contains a common core API with several basic classes implemented in Java, such as Position, Orientation, Transform, Mesh, Model, Shader, and others, which can be used for quick prototyping of VR apps. The repository also contains several VR projects ready to use as a development starting point. The projects range from a basic tutorial on how to compose 3D polygonal shapes to a full-scale demo of the J4Q API that implements an endless runner game in VR with controller interaction.

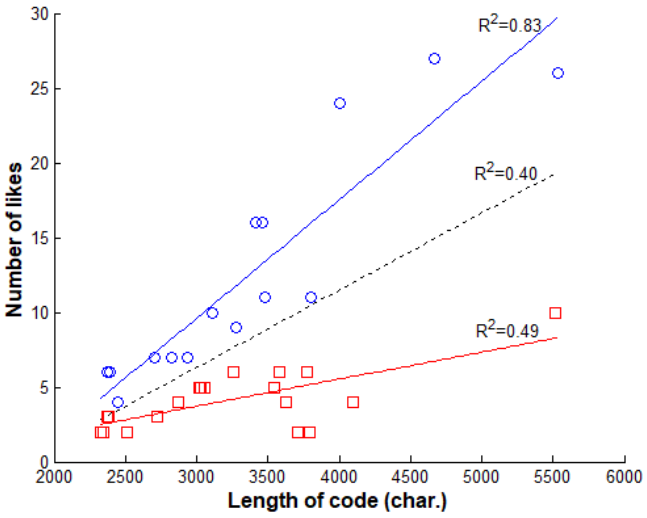


FIGURE III

PLOT OF THE DISTRIBUTION OF STUDENT 3D ARTWORKS BASED ON THEIR EFFORT (LENGTH OF CODE) AND AESTHETIC VALUE (NUM. OF LIKES)

EMPIRICAL EVIDENCE AND DISCUSSION

The proposed framework was designed iteratively in four cycles based on our observations and student feedback from the undergraduate-level class at the University of Florida, 'Wearables and Mobile App Development', during the Fall semesters of 2019-2023. Although we have not yet designed a formal study to assess this framework quantitatively, we briefly list some of our observations below:

- The students repeatedly reported that they learned and understood programming better by making VR apps compared to traditional CS courses on programming.
- Several students felt they became more confident users of 3D modeling and professional game design tools after learning about their 'under the hood' details.
- The students better grasped advanced CS concepts, such as optimization and algorithmic complexity, when working with meshes and shaders in our framework.
- The students creatively used the coordinate system transform commands to compose artful 3D models.

Figure 3 illustrates the relationship between the coding effort and the aesthetic outcome of 3D models composed using the J4Q API. We noticed two distinct patterns: 1) peers upvoted the artworks with better aesthetics, and their popularity increased by the coding effort (shown in blue), 2) designs that lacked artistic value received fewer votes, but they also increased with effort (in red). This observation was evaluated by the coef. of determination (R^2) and using K-means (K=2) along with linear regression to fit the lines shown in Fig. 3. The top row of Fig. 1 shows examples of upvoted models, and the bottom row shows examples of less popular designs.

CONCLUSION AND FUTURE DIRECTIONS

This paper demonstrated how the process of making or modifying a small-scale VR game engine can be used as an experiential learning platform for teaching technical concepts that are typically left unexplored when using premade game engines. We provided examples of educational activities that can be used along with our open-source implementation introduced in this paper. The proposed framework can be used in DAS curricula for training future technical artists, special effects artists, animators, and other creative media professionals. In the future, we would like to identify K-12 common core standards and design a study to assess if the proposed framework can also be used in secondary education.

REFERENCES

[1] Perlis, Alan J. "The Computer in the University." In Greenberger, Martin (ed.), 1962, *Management and The Computer of the Future*, New York and London: John Willey and Sons, pp.180-217.

[2] Buckingham, R. A., 1965. "The computer in the university." *The Computer Journal* 8(1), pp.1-7.

[3] Wang, V. and Wang, D. 2021. "The Impact of the Increasing Popularity of Digital Art on the Current Job Market for Artists." *Art and Design Review* 9, pp. 242-253.

[4] "Occupational Outlook Handbook: Arts and Design Occupations", U.S. Bureau of Labor Statistics, 2022. https://www.bls.gov/ooh/arts-and-design/home.htm. Web. Accessed: January 3, 2023.

[5] Piteira, Martinha and Costa, Carlos 2013. "Learning computer programming: study of difficulties in learning programming". In *Proceedings of the 2013 International Conference on Information Systems and Design of Communication (ISDOC '13)*, New York, NY: Association for Computing Machinery, pp. 75–80.

[6] Friend, M., Matthews, M., Winter, V., Love, B., Moisset, D. and Goodwin, I. 2018. "Bricklayer: Elementary Students Learn Math through Programming and Art." In *Proceedings of the 49th ACM Technical Symposium on Computer Science Education (SIGCSE '18)*, New York, NY: Association for Computing Machinery, pp. 628–633.

[7] Reas, Casey and Fry, Ben 2007. *Processing: A Programming Handbook for Visual Designers and Artists*, Cambridge, Massachusetts: MIT Press.

[8] Pinto, M. and Terroso, T. "Learning Computer Programming: A Gamified Approach." In Simões, Alberto and Silva, João Carlos (eds.), 2022, *Third International Computer Programming Education Conference (ICPEC 2022)*, Dagstuhl: Schloss Dagstuhl – Leibniz-Zentrum für Informatik. pp. 11:1-11:8.

[9] Venter, Marisa 2020. "Gamification in STEM programming courses: State of the art", In Proceedings of the 2020 IEEE Global Engineering Education Conference (EDUCON), pp. 859-866.

[10] "SCRATCH", Massachusetts Institute of Technology. https://scratch.mit.edu/. Web. Accessed: January 3, 2023.

[11] Kafle, Sachin 2019. *Learning Python by Building Games: A Beginner's Guide to Python Programming and Game Development*, Cambridge: Packt Publishing Ltd.

[12] Kavanagh, S., Luxton-Reilly, A., Wuensche, B. and Plimmer, B. 2017. "A systematic review of Virtual Reality in education". *Themes in Science and Technology Education* 10(2), pp. 85-119.

[13] Papert, Seymour. "What is Logo? Who Needs It?", In 1999, *Logo Philosophy and Implementation*, Logo Computer Systems, Inc., pp. IV-XVI.

[14] Flores, N., Paiva, A. C., & Cruz, N. 2020. "Teaching Software Engineering Topics Through Pedagogical Game Design Patterns: An Empirical Study." Information, 11(3), 153.

[15] Dickson, P. E., Block, J. E., Echevarria, G. N., & Keenan, K. C. 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, pp. 70-75.

[16] Dickson, Paul E. 2015. "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 (ITiCSE '15). New York, NY: Association for Computing Machinery, pp. 75–80.

[17] González-Zamar, M. D., & Abad-Segura, E. 2020. "Implications of virtual reality in arts education: Research analysis in the context of higher education." Education Sciences, 10(9), 225.