Project Elements: A computational entity-component-system in a scene-graph pythonic framework, for a neural, geometric computer graphics curriculum

G. Papagiannakis\textsuperscript{1,2,5} M. Kamarianakis\textsuperscript{1,2,5} A. Protopsaltis\textsuperscript{3,5} D. Angelis\textsuperscript{1,2,5} P. Zikas\textsuperscript{4,5}

\textsuperscript{1} Foundation of Technology Hellas - ICS, Heraklion, Greece
\textsuperscript{2} University of Crete, Dept. of Computer Science, Heraklion, Greece
\textsuperscript{3} University of Western Macedonia, Dept. of Electrical and Computer Engineering, Kozani, Greece
\textsuperscript{4} University of Geneva, Dept. of Computer Science, Geneva, Switzerland
\textsuperscript{5} ORamaVR SA, Geneva, Switzerland and Heraklion, Greece

Abstract
We present the Elements project, a lightweight, open-source, computational science and computer graphics (CG) framework, tailored for educational needs, that offers, for the first time, the advantages of an Entity-Component-System (ECS) along with the rapid prototyping convenience of a Scenegraph-based pythonic framework. This novelty allows advances in the teaching of CG: from heterogeneous directed acyclic graphs and depth-first traversals, to animation, skinning, geometric algebra and shader-based components rendered via unique systems all the way to their representation as graph neural networks for 3D scientific visualization. Taking advantage of the unique ECS in a Scenegraph underlying system, this project aims to bridge CG curricula and modern game engines (MGEs), that are based on the same approach but often present these notions in a black-box approach. It is designed to actively utilize software design patterns, under an extensible open-source approach. Although Elements provides a modern (i.e., shader-based as opposed to fixed-function OpenGL), simple to program approach with Jupyter notebooks and unit-tests, its CG pipeline is not black-box, exposing for teaching for the first time unique challenging scientific, visual and neural computing concepts.

CCS Concepts
- Social and professional topics → Computer science education; Software engineering education;

1. Introduction
Computer Graphics is a challenging teaching subject [SWLR17, MS23], as it borrows knowledge from multiple computational scientific areas [BWF\textsuperscript{*}17] such as mathematics, biology, physics, computer science and specifically software design, data structures and GPU programming through shader-based APIs. Along with the multiple pedagogical approaches, there exist a variety of tools and frameworks [TRK17, AGCCFGVP18, Mil14, SWLR19, BSP17, PB21, WHS\textsuperscript{*}19, WLD\textsuperscript{*}22, TRK17] that facilitate CG development; however, only a small subset is oriented towards teaching a hands-on, programming and assignment-based approach of a part of or the complete modern graphics pipeline, from shader-based visualisation all the way towards neural computing. An approach that seems to yield positive feedback from the students is the use of notebooks [PB21], usually based on WebGL, that help visualize concepts such as lighting, shadows and textures in an interactive way [WHS\textsuperscript{*}19]. Such tools offer limited functionality as they focus on specific CG aspects, detached from the CG pipeline [WLD\textsuperscript{*}22] such as the GLSL shaders [TRK17]. In such approaches, students may focus and learn in depth how a particular task is performed or how specific parameters affect the final rendered scene, but they usually have trouble comprehending how these operations are interconnected into a single pipeline. Another strategy to teach the entire operation of the OpenGL pipeline is via the employment of frameworks [AGCCFGVP18, Mil14, BSP17]. These packages usually contain examples and helper functions that perform batch-calling of simpler OpenGL functions, based on their functionality, e.g., shader initialization or VAO/VBO creation, thus facilitating students understanding without exposing them to low-level code, until they are ready to handle or tamper with it. Such frameworks are usually written in C++/C# and are suitable both for teaching as well as small or intermediate projects. A main drawback of them is that several CG pipeline operations are treated as black-boxes, thus not providing the practical knowledge of all stages of the rendering pipeline to students, like the calculation of the global transformation matrix of a 3D mesh in a scenegraph. Furthermore, these frameworks are not extensible in a simple, straightforward and open-source way towards state-of-the-art python-based neural...
computing for geometric deep learning frameworks such as graph neural networks.

1.1. A rising tide: ECS and SceneGraphs

The management of interactive scenes in MGEs and graphics systems is based on a hierarchical data structure, the scenegraph, a heterogeneous, directed acyclic graph, that is traversed to render a frame [RH94]. The scenegraph contains all the data required to replicate the scene, including among others, geometry, materials, camera specifications and light information. All graph nodes inherit properties from their ancestors and the object mesh data of a specific node are stored on child leaf nodes. The nodes are usually referred to as gameobjects, actors or plain objects in MGEs, whereas the related data stored within them are usually denoted as Components. SceneGraph edges denote connectivity and hierarchy; different traversals ensure initialisation, update, culling and rendering of all scene graph nodes.

Entity-Component-System is an architectural pattern, mainly used in 3D applications and game development [Nys14, PQKN17], that decouples data from behavior, simplifying the development. It heavily relies on data-oriented-design and composition, where created entities are assigned independent components, in contrast with object-oriented design, where such components would be inherited from base classes. This architecture enables a) improved performance in applications with many objects (e.g. physics based simulations) and b) improved maintenance and understanding of the application’s objects.

The ECS model has gained a lot of attention in various game frameworks (EnTT, https://github.com/skypjack/entt), [RH19] and a variation of it is in the core of most MGEs. For example, Unity is currently restructuring its core-engine, towards adopting the DOTS (acronym for Data Oriented Technology Stack) system (https://unity.com/dots) that features an ECS architecture, to tackle violations caused by previous data-oriented programming principles and achieve better fps performance in complex scenes. In our work, we provide a unique core ECS in a scenegraph architecture, in order to teach several key CG concepts in a data-oriented approach that facilitates connectivity with neural computing and geometric deep learning.

1.2. Current approaches in modern CG teaching

To enable a CG hands-on programming approach, students are usually given a framework and initial pipeline skeleton examples, upon which they may build up their first CG applications. Such approaches utilize a MGE, or a C/C++ framework.

MGEs, such as Unity and Unreal engine, are high-level tools that provide excellent visualizations of content using all CG principles, but most low-level details of the CG pipeline are kept hidden in a black-box. For instance, any Unity programmer may easily create CG content, such as a shadowed cube with no knowledge in lighting or shadow algorithms, or animate a simple model without having to deal with the frame (linear or spherical) interpolation process and the calculation of the world transformation matrix, or rotate an object without knowing the details of Euler angles or quaternions.

In that frame, as the learning objectives of all CG courses include the low-level experimentation of students with all basic CG principles, CG professors have suspended the use of game engines in undergraduate CG courses. On the other hand, game engines are widely used in graduate CG courses, where students have already mastered CG principles well.

In many CG curricula, professors have opted for the use of C/C++ CG frameworks or packages, prioritizing the learning of basic principles of the CG pipeline (rendering, lighting, shading). Although frameworks are often hierarchically structured and code exploration is, in some cases, possible at some levels, the complexity of such frameworks is great, hindering swift prototyping and connection with rapidly growing scientific domains. Furthermore, a fully educational-tailored experience at all levels is still not feasible, and state-of-the-art concepts, such as ECS on a scenegraph and design patterns are not utilized yet.

1.3. Our pythonic, ECS in a SceneGraph, approach: ECSS

In this paper, we introduce the Elements project, that aims to provide the power of the Entity-Component-System (ECS) in a SceneGraph (ECSS) pythonic (i.e., using latest python 3 language features and software design patterns) framework, suitable to be used in the context of a CG curriculum with extensions to scientific, neural computing and geometric deep learning.

As a central idea, we have implemented a software-design-pattern based scenegraph, combined with an Entity Component System (ECS) [Bib02] principle, which dictates a data-oriented rather than an object-oriented programming approach. By opting to ECS, we were able to serve the main approach that MGEs abide to and grants them the ability to effectively manage even extremely complicated scenes.

In our ECSS, the functionality of our scenegraph is provided via the Composite pattern, our Entities are just Grouping nodes with an ID, our Components contain only data that can be easily extended using the decorator pattern, and all functionality (rendering, update etc.) is only performed in our Systems that are traverse the scenegraph of Entities and Components via the visitor pattern. Finally, events and messages in our ECSS can easily be handled using the observer pattern.

Although scenegraphs, shader-based GPU programming and ECS are well known principles in the CG community, the existence of a framework that incorporates all of them and exposes them as easily accessed notebooks for scientific and neural computing has not appeared in the bibliography so far, to the best of our knowledge. Included in Elements (see Fig. 1), the pyECSS (acronym for python ECS on Scenegraphs), is a unit-test python package, that, for the first time, incorporates both of these principles, in a comprehensive way, suitable for teaching to CG students.

The pyGLV (acronym for computer Graphics for deep Learning and scientific Visualization in python) package of Elements exploits the pyECSS benefits and provides the necessary graphics algorithms to allow the creation and visualization of 3D scenes via the OpenGL and GLSL APIs and languages. The use of simple functions allows the creation of Entities (scene root and objects)
and their Components (geometry, camera, lights, transformation) of objects in the 3D scene. CG programmers can quickly dive into the concept of a scenegraph, an approach that will help them easily adapt to the pipeline of MGEs. The built-in Systems (TransformationSystem, CameraSystem) and the shader functions prove to be essential for beginners, as they help them grasp the CG principles without initially worrying about low-level implementation details.

Lastly, the Elements’ pyEEL (acronym for Explore - Experiment - Learn using python) repository holds various jupyter notebooks, suitable for both beginners and experienced programmers. Our aim is to evolve this repository to a knowledge-hub, a place where tutorial-like notes will help the reader comprehend and exploit the pyECSS and/or the pyGLV package to their full extent. As many research frameworks in diverse scientific and neural domains are python based, we envision to further contribute in such scientific areas, as Elements is best suited for data visualization, immersive analytics and geometric deep learning with graph neural networks (GNNs).

1.4. Benefits of using the Elements project in CG curricula

Below is a list of the benefits from using Elements in a CG class, both as an instructor as well as a student.

1. ECS in a Scenegraph (ECSS). Teaching the benefits of ECS when applied in a scenegraph will allow the students to understand the main CG concepts using a rendering pipeline that is very similar to the ones utilized by MGEs, which can support rendering million of objects in real-time. This will assist students on organizing and designing a scene or on incorporating new features. As individual Systems can work autonomously and in parallel with other Systems, a faster rendering can be obtained. This ECSS approach can also be followed in other domains besides CG, or for other CG tasks besides scene rendering, like physics computations. To this end, ECSS functionality is encapsulated in the standalone pyECSS python package.

2. A python-based, visual computing framework based exclusively on software design-patterns. Python is a language that is both ideal for novice and proficient CG programmers as well as becoming a de facto standard in neural computing. Its versatility comes from the rapidly growing variety of developed scientific libraries and frameworks, by its large active community.

Figure 1: The Elements Project: pyECSS, pyGLV & pyEEL.

2. The key components of the Elements project

2.1. The pyECSS package

The pyECSS python package, based on pure software design patterns, features a plain but powerful CG architecture with Entities, Components and Systems in a Scenegraph. A class diagram of the pyECSS package and their in-between dependencies is shown in Fig. 2. The pyECSS package, after a simple installation via PyPi, allows the programmer to easily create a scenegraph using Entities and Components as nodes. A simple example is shown in Listing 1, where several entities are created and interconnected via parent-child relationships and component nodes are added to each entity.

Listing 1: Scenegraph creation code snippet where we add and connect entities and components.
Within the developed ECSS framework entities act as unique identifiers, components contain the associated data and systems are classes that operate on both of them, performing various tasks, such as updating the position of an entity based on the root or passing a visible object to the vertex shader.

The advantage of using an ECSS is that it allows developers to build flexible, modular systems that can easily be extended and modified. It also makes it easier to reason about the interactions between different objects in the scene, since each object is represented by its own set of components. Such ECSS architectures are particularly useful in CG and gaming applications, where scenes with large number of objects, each with complex behaviours, that need to be updated and rendered in real-time, is a common issue.

2.2. The pyGLV package

The pyGLV python package, based on pure software design patterns, utilizes the previous ECSS architecture, via the pyECSS package. Its main aim is to showcase the applicability of basic, cross-platform, OpenGL-based, real-time computer graphics, in the fields of scientific visualization and geometric deep learning. A class diagram of the pyGLV package and their in-between dependencies is shown in Fig. 4.

The pyGLV contains the implementation of many built-in components, that are used to setup CG scenegraphs, using the pyECSS package.

- **BasicTransform.** A component that stores the objects’ coordinate system with respect to its parent entity, in the form of a product of a translation, rotation and scaling matrix.
- **Camera.** A component storing the camera setup information in the form of a view matrix; it can be generated via the ortho or perspective functions, contained in the pyECSS.utilities package.
- **RenderMesh.** A component that stores the basic geometry of an entity, such as the vertex positions and face indices of an object. It may also contain the colour of the object or the colour of each vertex, as well as the normals of each face triangle.
- **VertexArray.** A component used to store the vertex array object (V AO) and vertex buffer object (VBO) for entities that will be passed to the vertex shader.
- **Shader.** A component that stores the data required for an OpenGL-GLSL Shader; it contains basic vertex and fragment shaders ready to be used.

The code segment in Listing 1, depicts the attachment of BasicTransform, RenderMesh, a VertexArray and Camera components on various entities of the scene. Depending on the situation, new components may be created to store different types of data, while existing components may be decorated to allow the storage of additional data of the same or equivalent representation formats, such as store transformation data in alternative algebraic forms such as dual-quaternions.

Furthermore, the pyGLV package contains the implementation of a set of systems, responsible to digest the scenegraph and apply various tasks of the CG pipeline, such as the evaluation of the

---

**Figure 2:** pyECSS class diagram

**Figure 3:** A simple scenegraph based on ECS. Circular nodes correspond to Entities while rectangular correspond to Components. The directed edges imply a parent-to-child relationship. Transform system evaluates the local-to-world matrix for all entities the root-to-camera is determined via the Camera System. Transformation data for Entity1 is provided in matrix form, whereas CGA multivectors are employed for Entity2; $e^\infty_0, e^o_0$ are basis multivectors, $t, b$ are the translation vector and the rotation axis expressed as multivectors, $\phi$ is the rotation angle, $d$ is the dilation factor and $e, \wedge$ denote the exponential and wedge product functions.

\[ T = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \]

\[ mT^R = D, \text{where} \]

© 2023 The Authors.
Proceedings published by Eurographics - The European Association for Computer Graphics.
model-to-world and root-to-camera matrices, through scenegraph traversals.

1. **TransformSystem**: A system traversing the scenegraph, evaluating the local-to-world matrix for each object-entity. This is accomplished by multiplying all Transform component matrices, starting from a given node-object and ascending the graph to its root-node (see Fig. 3); the order of matrix multiplication is the same to the one taught in the theoretical CG course; the first transformation matrix is the right-most in this product.

2. **CameraSystem**: A system evaluating the root-to-camera matrix in a CG scenegraph. This crucial for the CG pipeline matrix is evaluated via this system, by identifying the single node with a camera component and returning the inverse of the model-to-world matrix for that entity (see Fig. 3).

3. **InitGLShaderSystem**: A system used to make data initialization and start the CG engine, outside the main rendering loop.

4. **RenderGLShaderSystem**: A system responsible for the main GPU rendering. When both a VertexArray and a Shader component are encountered under the same entity, the system passes the VAO to the GPU and visualizes it on screen.

Furthermore, it is possible to decorate these systems which will allow the digestion of decorated or recently introduced components, depending on the aimed application functionality. As an example, one could create a texture component along with the respective system, that would apply the texture to a specific object/entity.

### 2.3. The pyEEL repository

The python-Explore-Experiment-Learn (pyELL) repository contains several python-based jupyter notebooks, that act as a knowledge hub, showcasing basic, cross-platform, scientific computing and OpenGL-based real-time computer graphics with applications to scientific visualization and deep learning. We envision to extend this repository to a portal for both beginner and veteran programmers, that want to learn the basics or advanced techniques in various domains/packages, including python, numpy, git, matplotlib, ECS, Geometric Algebra, Graph Neural Networks (GNNs), etc.

### 3. Teaching CG with Elements at (under)graduate levels

In this section we present how Elements has already been utilized in several introductory and advanced CG courses of undergraduate/graduate students. The Elements’ features are unfolded sequentially, according to the syllabus of the specific CG course, allowing students to gradually apply in practice the entire set of theoretical knowledge they acquire.

After the introductory lesson to CG and its importance (week 1), students are taught the basic mathematics involving homogeneous coordinates, rigid body transformations, projections and viewing matrices (weeks 2 & 3). Since enrolled students are at least in their 3rd year, this is simply an extension of their knowledge of linear and vector algebra. In week 4, they are introduced to graphics programming during the 1st Lab, involving the setup of the Elements project and experimenting with basic mathematical functions. After learning more on geometry and polygonal modeling, as well as scene management (week 5), the 1st assignment is released (see Section 5) and the 2nd lab takes place, where the initial Elements examples are explained. As the course progresses, students learn about hardware lighting & shading, advanced GLSL (week 6), textures and basic animation (week 7). The 3rd lab (week 7) then exposes students to the respective Elements examples that demonstrate the Blinn-Phong lighting algorithm, texture loading, object-importing and rigid-skinned model animation (eventually understanding all principles behind a scene such as Fig. 5). The relative 2nd assignment is released on week 8 (see Section 5).

The rest of the course unfolds without using Elements, as students must also get acquainted with a MGE such as Unity and even more complex frameworks. For completeness, we mention the remaining topics of the CG curriculum: Visible surface determination and real-time shadows are taught at week 8, followed by ray-tracing and CG in 3D games (week 9). The remaining weeks 10-13 involve Unity lectures & tutorials, a lecture on VR & AR techniques, along with a hands-on lecture with VR & AR head-mounted displays. A Unity-based assignment is released on week 10.

### 4. Elements framework student assignments

In this section, we describe few key graduate and undergraduate assignments that were developed using the Elements project.

#### 4.1. Skinned Animated Objects in Elements

In this assignment, students extended the basic mesh component to a SkinnedMesh component and created a respective system to allow animation of rigged animated models in Elements. The new component allows the storage of skinned 3D model data, which are loaded via the pyassimp package. The corresponding system digests such components and applies the animation equation to the skinned model between two keyframes (see Fig. 6).
independently and in parallel (see Fig. 3).

...programming effort, since systems digesting such components work...

...multivectors, quaternions, dual-quaternions, multivectors) to define the...

...the programmer may use diverse algebraic formats (matrices, ...

...are an evolution step of current representation forms [KCL∗22].

...such transformations (translations, rotations and dilations), which are an evolution step of current representation forms [KCL∗22].

...As such, the use of GA-based forms (motors, rotors and dilators)...
operating room”. The auto-creation of such scenes is not straightforward, and should be decomposed in a set of simpler tasks, such as the generation of 3D objects and their spatial arrangement.

5. Using Elements in existing CG curricula

The Elements project has been employed as a teaching material for a graduate (CS-553 Interactive Computer Graphics) and an undergraduate (CS-358 Computer Graphics) CG course at the University of Crete (UoC), Greece as well as for the undergraduate (ECE-34 Computer Graphics) CG course at the University of Western Macedonia (UoWM), Greece. The number of students actively participated in these assignments, using the Elements project, were 28, 24 and 24 respectively.

There were two assignments within the undergraduate course taught in UoC that regarded the use of the Elements project. The first one, given on week 5, regarded simple tasks such as using the built-in functions for transformations (translate, rotate and dilate), for camera-related functions (lookat and perspective), as well as generation of models (polygons and sphere) in a constructive way. The second one was given on week 8 and involved evaluation of normals via model processing, scene manipulation via a GUI system and implementation of the Blinn-Phong model. Although there were 4 less participants in the second assignment (28 in the first and 24 in the second), the grades indicate that students quickly gain expertise on the Elements project, just a few weeks after, and managed to perform better, even in more demanding subjects (see Fig. 8, comparing orange and blue bars).

In UoWM, the traditional approach of teaching OpenGL in C/C++, without the use of a specific framework, was augmented with Elements. Students, having already acquired knowledge on basic OpenGL programming, attended three 2-hour laboratory lectures in order to get acquainted with Elements and were given a single, optional, related assignment, aiming to evaluate and compare students’ understanding on basic and advanced CG pipeline stages. The assignment involved the built a 3D scene graph, along with the generation of custom polygonal 3D models, a GUI system that allows transformations and animations applied on a single or group of 3D objects, and the Phong illumination model through GPU scripting. Even after such a short introduction, the 24 students that participated managed to easily adapt. Course evaluations highlighted the direct relation of Elements with the taught theory, compared to the traditional OpenGL teaching approach; their grades, shown as gray bars in Fig. 8, are quite above average.

6. Performance

Using Elements, the rendering of moderately complex scenes is possible in real-time; in our tests, we managed to load up to 150 objects, each one of approximately 2900 vertices, without dropping below 60fps (tested with a MacBook Pro, 2.6 GHz 6-Core Intel Core i7, 16 GB 2667 MHz DDR4, Intel UHD Graphics 630 1536 MB). All parts of the implementation are intentionally written in python, to promote student understanding and allow the fast creation of proof-of-concept examples. Our primary goal when designing Elements was the maximization of the educational impact rather than the rendering performance.

7. Conclusions and Future Work

In this work, we have presented Elements, a novel open-source pythonic entity-component-systems in a scenegraph framework, suitable for scientific, visual as well as neural computing. The python packages comprising Elements: pyECSS, pyGLV and pyEEEL include the basic implementation of such an approach along with useful examples that may be used to quickly introduce even inexperienced CG programmers to main CG-programming principles and approaches. Despite its simplicity, it remains white-box, allowing users to dive in the graphics pipeline and tamper with any CG pipeline step. Empowered by python’s rapid prototyping and developing ability, users may implement new or decorated components and/or systems to extend the functionalities offered by Elements. The impact of Elements’ features (current and future) on various scientific domains/packages are demonstrated in jupyter notebooks, in the pyEEEL repository, towards creating a learning hub for both novice and intermediate developers.

Taking advantage of the ECSS underlying system, this project aims to bridge CG curricula and MGEs, that are based on the same approach. This will prepare students, that develop their CG knowledge, to take the next step to game engines. The bridging will also enable rapid prototyping in Elements and a subsequent easy port to game engines, to further evaluate its performance or other metrics, not accessible in a python environment.

Work in progress involves the integration of the Elements project
with open source packages, such as OpenXR, that will allow the 3D scene to be output in virtual or augmented reality head-mounted displays, which will in turn allow enhanced data visualization and, ultimately, immersive analytics. Furthermore, the connection with modern Unity-based frameworks such as [ZPL*23] are also actively explored. As major neural deep learning frameworks are python-based, we aim to provide more powerful and versatile ways of scientific visualization (input/output, to/from). Furthermore, we will strengthen the connection of Elements with GNN-based geometric deep learning techniques towards performing more complex tasks, such as generating a new ECSS.

A subjective grade-based comparison with previous years is not feasible, as different frameworks, assignments and syllabus were used back then. However, we will keep on monitoring the benefits that Elements can bring to CG curricula that adopt its use and conduct further evaluations on student reactions as well as randomized control trials, towards highlighting the added pedagogical values of using the proposed project instead of classical C/C++ or other frameworks.

Acknowledgements
We would like to thank Ioanna Patsoura and Mike Kentros for their contribution, related to the assignments presented in this paper.

Code Availability
Elements is available via open-source at https://papagiannakis.github.io/Elements/.

References