

Conceptual architectural model of an instructor-oriented Symbolic-to-Visual pipeline for bulk production of individualized surface flux learning materials*

Halyna Tuluchenko^{1,*†}, Yuliia Pershyna^{1,†} and Vitaliy Mezhuhev^{2,†}

¹ National Technical University "Kharkiv Polytechnic Institute", Kyrpychova 2, 61002, Kharkiv, Ukraine

² FH JOANNEUM University of Applied Sciences, Institute of Industrial Management, Werk-VI-Straße 46, 8605, Kapfenberg, Austria

Abstract

This paper presents a conceptual architectural model of an instructor-oriented Symbolic-to-Visual pipeline designed for the bulk production of individualized learning materials on the flux of a vector field through a plane. The model is developed as a component of an evolving vector calculus learning environment and integrates real-time symbolic validation, asynchronous procedural 3D rendering, and deterministic generation of synchronized computational (CSV) and visual (MP4) components within a single environment. It enables instructors to produce validated task packages without direct interaction with low-level scripting, while ensuring mathematical invariance of the vector field description and perceptual stability of the 3D scene.

Keywords

software module design, symbolic-to-visual data pipeline, automated authoring tools, Python, ManimCE, procedural rendering, surface flux computation, Vector Calculus, learning environments

1. Introduction

The current development stage of digital learning environments in higher technical education requires a fundamental revision of approaches to designing the complete educational content creation lifecycle. This issue is particularly acute in mathematical disciplines such as Vector Calculus, where the understanding of fundamental concepts — such as computing the flux of a vector field through a surface — critically depends on the quality and dynamics of 3D visualization. Traditional approaches relying on static graphical materials no longer provide the required level of visual clarity, thereby stimulating the development of specialized software solutions. However, implementing an individualized approach to learning requires more than isolated demonstrations; it demands the scalable, bulk production of computational and visual learning materials. To build sustained academic authoring workflows for content development, educators require tools capable of automatically generating dozens of unique task variants comprising both computational and graphical components. While procedural animation frameworks such as ManimCE provide high-precision mathematical graphics, their command-line orientation creates a practical barrier for mathematics instructors. This gap motivates the development of intermediate architectural layers that translate instructor-defined mathematical expressions into synchronized computational and visual outputs. From an architectural perspective, a central challenge consists in the coordinated integration of three heterogeneous components — real-time symbolic computation, resource-

*ADT-ISMDCC 2026: Advanced Digital Technologies: Intelligent Systems, Modeling, and Data-Driven Control, September 22–24, 2026, Odesa, Ukraine

^{1*} Corresponding author.

[†] These authors contributed equally.

✉ Halyna.Tuluchenko@khp.edu.ua (H. Tuluchenko); Yuliia.Pershyna@khp.edu.ua (Yu. Pershyna); vitaliy.mezhuhev@fh-joanneum.at (V. Mezhuhev)

id 0000-0002-6196-540X (H. Tuluchenko); 0000-0002-4719-8195 (Yu. Pershyna); 0000-0002-9335-6131 (V. Mezhuhev)



© 2026 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

intensive procedural 3D rendering and a responsive graphical interface — into a single deterministic pipeline capable of bulk production of synchronized didactic packages. While separate solutions for asynchronous execution, symbolic validation and parameterized generation exist, their simultaneous synchronization under the combined constraints of mathematical invariance, perceptual stability of the 3D scene and zero-code interaction for the instructor remains insufficiently addressed for the domain of surface flux computation.

The objective of this study is to formulate, substantiate and experimentally validate a conceptual architectural model of an instructor-oriented Symbolic-to-Visual pipeline that enables the bulk production of individualized surface flux learning materials by integrating real-time symbolic validation with synchronized generation of task parameter sets (CSV) and corresponding visual components (MP4) intended for transfer to the student module, without requiring direct programming by the instructor.

To achieve the stated objective, this paper is structured as follows. Section 2 provides a review of relevant literature and existing approaches to automating educational material creation. Section 3 presents the architectural synthesis of the instructor-oriented Symbolic-to-Visual pipeline, focusing on component interaction, deterministic state management and process isolation. Section 4 addresses the geometric and cognitive modeling of the 3D flux scene. Section 5 describes the practical implementation of the system and demonstrates its functional capabilities. Section 6 presents a discussion of the obtained results, their significance, and limitations. Finally, Section 7 outlines the conclusions and prospects for further project development.

2. Related Works

The design and implementation of interactive educational software for advanced mathematics, specifically vector calculus, requires addressing a complex set of interdisciplinary challenges at the intersection of computer science, computational geometry, and cognitive didactics. Modern meta-analyses and systematic reviews demonstrate that integrating dynamic visual interventions into mathematics education significantly enhances conceptual understanding compared to the static illustrations [14, 14]. However, as noted by Lowe and Boucheix [14], educational efficacy directly depends on the validity and cognitive load management of the animation. In the context of multivariable calculus, the deployment of spatial visualization tools, including virtual and augmented reality (VR/AR) technologies, enables students to better interpret the geometric meaning of mathematical operations [14, 14].

In programmatic mathematical graphic generation, the Manim open-source framework (ManimCE), popularized by the 3Blue1Brown project [14], has emerged as a key engine. Its pedagogical value for visualizing complex STEM concepts and automating content creation is established across comprehensive reviews [14, 14]. Manim is actively applied to generate educational animations across diverse domains, ranging from mathematical physics [14] to parsing algorithms and data structures [14, 14]. The role of Manim-based systems in transforming abstract mathematical text into dynamic spatial models (the Symbolic-to-Visual pipeline concept) is outlined in [14], while recent approaches explore conversational agents and generative AI to automate script authoring [15].

As the Manim ecosystem evolved, several structural and graphical challenges inherent to its command-line execution model were identified and addressed. Integrating this computationally intensive rendering pipeline into desktop application GUIs required managing user interface responsiveness during long-running tasks. As demonstrated in the ANVIL framework [15], resolving these interface-synchronization requirements involves establishing asynchronous architectures, isolating rendering tasks within OS-level subprocesses, and utilizing structured data-interchange interfaces (e.g., JSON) to communicate with the symbolic core. Similarly, addressing optical occlusion

and perspective distortion of text labels during virtual 3D camera rotation established the necessity of dynamic topological constraints to preserve marker readability 15.

In the broader scientific visualization (SciVis) domain, general-purpose toolkits and platforms such as VTK 15, ParaView 15, and modular data-flow environments like Inviwo 15 serve as foundational standards for rendering complex volumetric data and physical phenomena. In educational contexts, specialized frameworks — such as the VisVisual and FlowVisual suites developed by C. Wang 15 — demonstrate the pedagogical efficacy of interactive flow field exploration. However, traditional SciVis ecosystems are engineered primarily for data-driven exploratory analysis, demanding steep learning curves and manual parameter configuration rather than automated batch generation of individualized educational assignments.

To address these limitations and streamline the preparation of visual didactic materials for higher mathematics — specifically for Vector Calculus courses — our recent line of research has focused on developing specialized software tools for automating candidate-specific 3D dynamic rendering and mathematical calculations. In particular, prior works 15, 15 substantiated the methodological frameworks and established approaches for animating vector fields via the Manim library, alongside introducing a gamified educational application ecosystem. The verification potential of such custom-built software modules in the mathematical training of engineers under global digitalization, along with their capacity to scale didactic material production, was validated in 15.

An evaluation of existing software solutions indicates that ready-to-use application environments integrating symbolic parsing, multi-variant 3D dynamic rendering, and automated calculation engines into a unified pipeline remain virtually non-existent. Due to the narrow domain specificity of vector calculus instruction, commercial development of such niche software is economically unviable. As a result, instructors are forced either to invest extensive effort into mastering complex programmatic animation or to rely on traditional static didactic materials.

3. Architectural Synthesis of the Instructor-Oriented Symbolic-to-Visual Pipeline

3.1. Ecosystem Integration and Functional Purpose of the Module

The design of the presented software module is based on the concept of a two-level digital learning ecosystem for vector calculus. This ecosystem synchronizes instructor and student activities through unified data packages and consists of two paired components:

- The instructor module (the focus of this study) — a high-level automated authoring tool for the batch creation of validated, individualized task variants without requiring direct programming;
- The student module (a prospective system component): an interactive learning environment that autonomously reads the instructor-generated parameters, computes reference analytical solutions for answer verification, and renders an individualized 3D scene as a cognitive reward for correctly solved tasks. This dynamic visualization offers additional insights into the spatial essence of the process, allowing students to directly observe the object of integration — something that remains purely abstract in traditional higher education approaches.

Within this ecosystem the instructor module has a clearly defined functional and mathematical purpose: rapid modeling and batch production of task variants for the computation of the flux of a

vector field through a plane. The automation pipeline is configured to work with surface integrals of the second kind: surface integrals of the second kind:

$$\iint_S \mathbf{F} \cdot \mathbf{n} dS, \quad (1)$$

where $\mathbf{F}$ is the vector field, S is the bounded region on the integration plane, and $\mathbf{n}$ is the unit normal vector.

The module is deliberately designed as a deterministic authoring tool rather than an autonomous solver or random parameter generator. Every component of a task remains under the full control of the instructor's pedagogical logic.

This design enables the instructor to adjust the didactic complexity flexibly by combining different types of vector fields (constant, polynomial, nonlinear) with arbitrary spatial orientations of the integration domain. Didactic package creation follows a strictly sequential, multi-stage process controlled by a system of interconnected graphical windows.

The two modules are connected through a data-driven pipeline. The output of the instructor module is a self-contained didactic package consisting of a CSV file with the numerical parameters of the task variants and a directory containing the corresponding 3D visualization videos (MP4). This package constitutes the sole data exchange channel between the modules: the student module imports the CSV file for automatic configuration of tasks and uses the video files as interactive support. In this way, the instructor module creates the complete didactic foundation that enables guided learning and automated verification within the student environment.

3.2. System Architecture and Component Interaction

The internal architecture is implemented in Python and follows the principle of loose coupling and deterministic state management. Interaction between the graphical interface (PyQt6), the analytical core (SymPy) and the procedural graphics engine (ManimCE) is realized exclusively through file-based data serialization and operating-system environment variables rather than direct in-memory object sharing. This design isolates the resource-intensive rendering process from the main UI thread. The main program window (FluxModule) acts as the central state controller. Transitions between computational and visualisation stages are strictly sequential: access to the advanced settings dialog remains blocked until analytical computation has completed successfully. The overall high-level architecture and the principal data and control flows are shown in Figure 1. The architecture deliberately separates concerns: symbolic validation remains inside the main process, while procedural 3D rendering is delegated to an external OS subprocess. Communication occurs only through a well-defined serialization bridge and environment variables, which reduces coupling and limits the impact of rendering failures on the interactive session. Reliability and controlled state progression are supported by three complementary mechanisms (see also Figure 2):

Context-aware gating via logical flags (`is_calculated`, `is_visualised`). These flags enforce sequential access in the main thread and prevent the graphics engine from receiving incomplete or null data. While flags alone cannot guarantee consistency under all asynchronous failure modes, they provide a lightweight and transparent means of maintaining workflow integrity under normal operating conditions.

JSON-based serialization bridge. All geometric parameters, colour settings and pre-computed symbolic results are written to an isolated configuration file under a context manager, ensuring deterministic hand-off and proper resource release. To ensure cross-platform stability, all file operations dynamically construct absolute paths (via `os.path` or `pathlib`), maintaining strict isolation between the temporary buffer zone, the project repository and the target library.

Subprocess isolation. Rendering is launched as an autonomous OS process; the configuration path is passed via an environment variable. This provides process-level isolation that goes beyond simple file exchange and protects the UI thread from rendering crashes or long-running computations. The configuration path is exported via the `FLUX_CONFIG_PATH` environment variable, which includes a local directory fallback mechanism to support autonomous script execution during independent testing.

The deterministic state transitions and flag-management rules are illustrated in Figure 2.

After successful final rendering, the main window sets the `is_visualised` flag, thereby unlocking the final saving action. Any subsequent modification of input fields resets both flags, ensuring that the system never operates on stale computational or visual results. This combination of architectural isolation, explicit state control and asynchronous dispatching forms the core of the proposed pipeline’s robustness.

3.3. User Interaction Workflow and Interface Organization

The external, event-driven layer of the system is organized as a sequential workflow controlled by the main window (Figure 3) and a set of supporting dialogs, key among which is the video parameter configuration dialog (Figure 4).

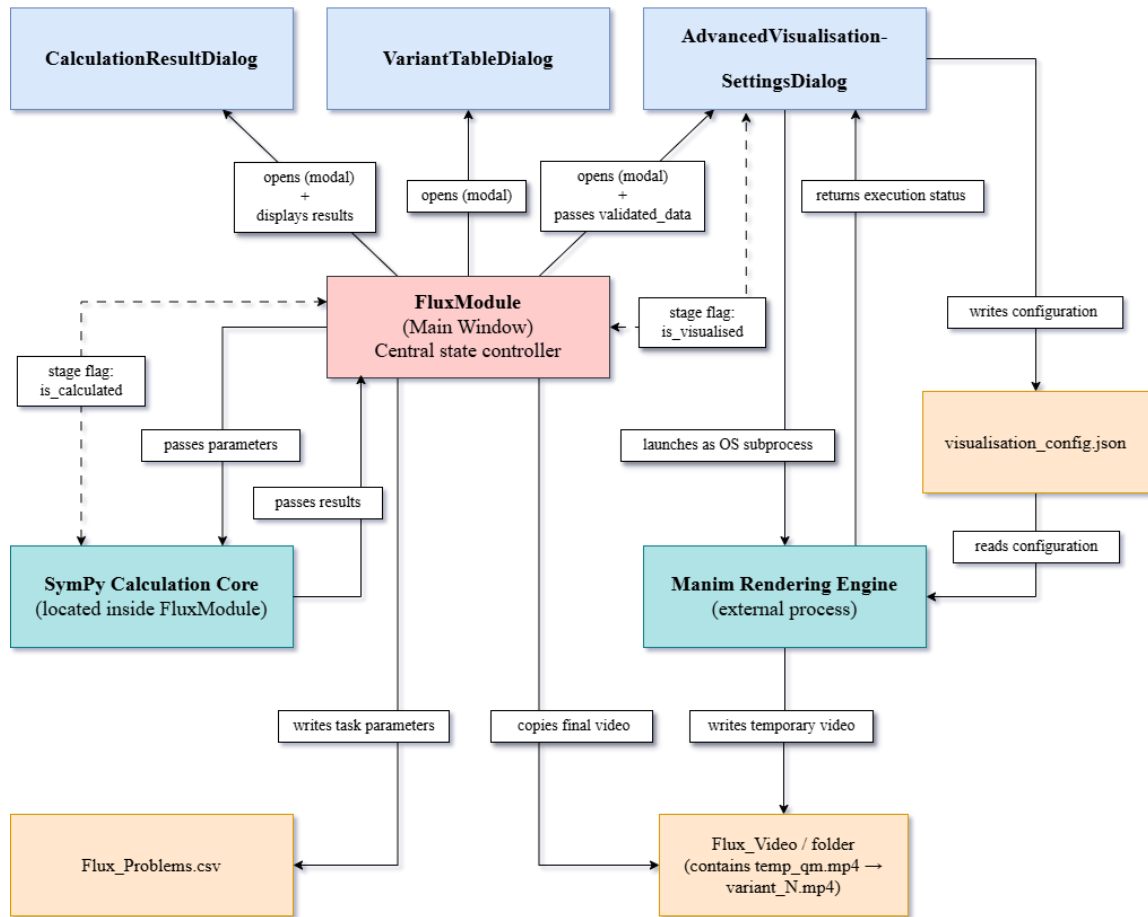


Figure 1. High-level architecture of the instructor-oriented Symbolic-to-Visual pipeline.

The workflow for creating a task variant comprises the following sequential states:

Workspace Initialization. At the initial stage, the instructor selects an existing folder (“Open Folder” button) or creates a new one (“Create New Folder” button) (Figure 3). The program

saves this path within the current session as the base repository, which subsequently handles the automatic distribution of media files and the formation of the general task registry.

Analytical Verification and Computation. After data entry the “Calculate” button initiates dynamic validation of the plane coefficients and field components, evaluates analytical computability of the surface integral, and displays the exact solution (or a message indicating its absence) in a modal window.

Spatial Scene Configuration and Media Asset Management. The “Visualisation” button (Figure 3) opens a dedicated window with advanced graphics parameters (Figure 4). Rendering is split into two asynchronous processes: a low-resolution draft preview (“Preview Video” button) and final medium-quality rendering (“Save Video” button). Both store temporary files in an isolated buffer zone.

Incremental Saving and Variant Library Management. Upon clicking “Add Variant” the system appends the numerical parameters to the CSV file and transfers the higher-resolution video to the target directory under a unique variant index. Crucially, the computed flux value (the exact numerical answer) is intentionally omitted from the generated CSV file, ensuring that students receive only the task parameters for independent solution. Before appending new records, the system verifies variant index uniqueness to prevent duplicate entries and re-sorts the database to preserve sequential visual ordering. An integrated table view (“View Table” button) allows continuous auditing of the populated database.

The sequential arrangement of controls and the modal nature of the dialogs reinforce the same state-transition rules that are enforced at the architectural level.

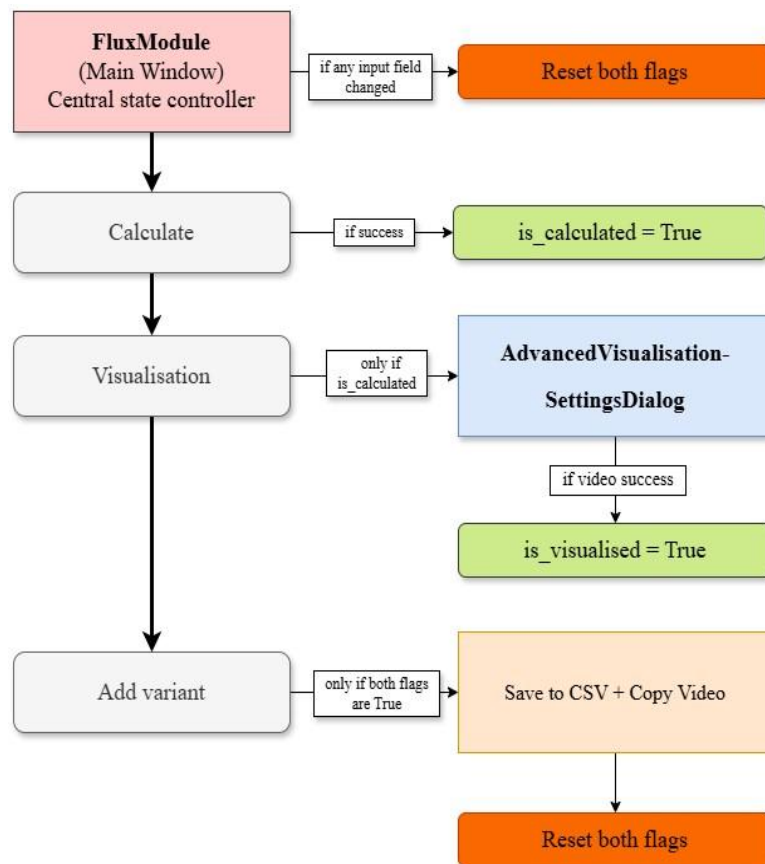


Figure 2. State control flow and flag management in the instructor module.

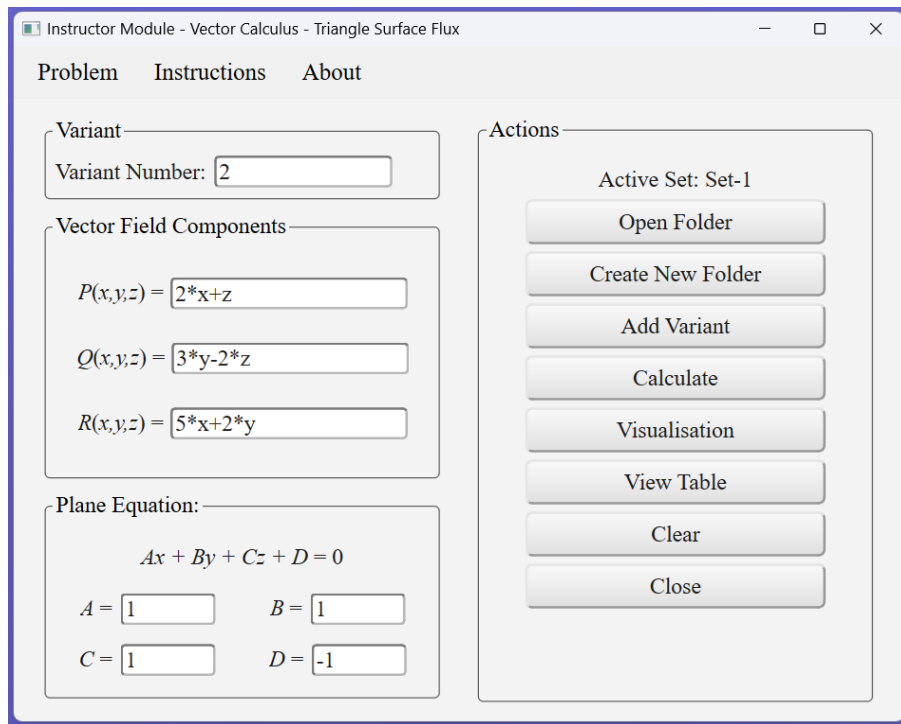


Figure 3: Main window of the instructor module's graphical user interface for entering parameters and managing the variant library.

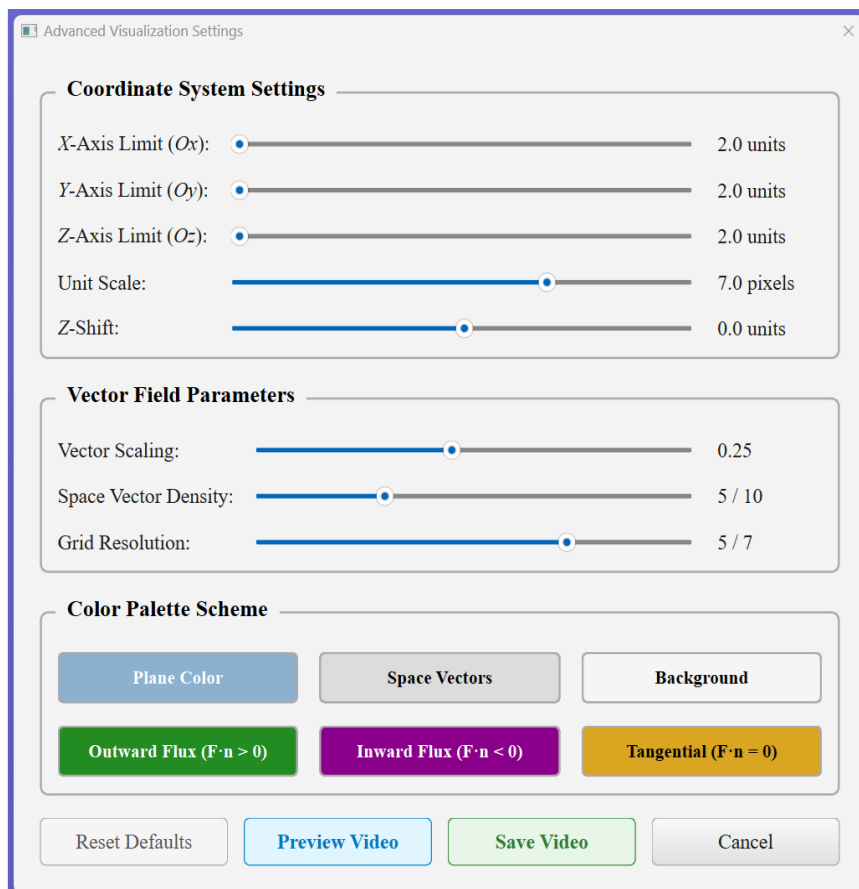


Figure 4: Dialog window for advanced visualization settings, configuring 3D scene parameters, and launching the rendering process.

4. Geometric and Cognitive Modeling of the 3D Flux Scene

Visualizing three-dimensional vector fields and surface fluxes is complicated by two interrelated problems: optical occlusion and cognitive overload of the scene. A dense spatial grid of vectors quickly produces visual chaos in which the geometry of the studied surface is lost. To address these issues the module implements a combination of layered geometric modeling and selective perceptual optimization within the ManimCE engine.

4.1. Layered Representation of the Vector Field: Global Spatial and Local Planar Components

A mathematically justified solution to scene clutter is the spatial and functional separation of the vector field into two independent layers (`space_vectors` and `plane_vectors`):

Background (Global) Spatial Vector Field. This layer is generated by an optimized `ArrowVectorField` and conveys the overall topology and flow trends throughout the examined volume. Grid density is controlled by a slider in the range from 3 to 10 (default value is 5), which sets the number of sample points along each edge of the bounding parallelepiped. The total number of background vectors therefore grows cubically. To limit occlusion at higher densities the arrows are rendered with reduced stroke width (1.8) and opacity (0.5).

Local (Target) Planar Vector Field. This layer constitutes the primary didactic element: it visualizes the direct interaction of the field with the triangular integration domain. An isoparametric grid based on Lagrange nodes of a triangular finite element is employed.

To correctly position geometric objects in the ManimCE engine, the physical coordinate system of the mathematical problem is first mapped to the internal three-dimensional coordinates of the virtual scene. The coordinates of the plane's intersections with the axes ($x = -D/A$, $y = -D/B$, $z = -D/C$) are transformed into scene vector objects p_1 , p_2 , p_3 using the built-in mapping method:

```
p1 = axes.c2p(-D / A, 0, 0)
```

```
p2 = axes.c2p(0, -D / B, 0)
```

```
p3 = axes.c2p(0, 0, -D / C)
```

Based on the obtained reference points p_1 , p_2 , p_3 and a specified density level, the barycentric coordinates of the triangular nodes are computed, where weights (w_1, w_2, w_3) are derived from integer indices satisfying the condition $i + j + k = \text{density} - 1$. The Cartesian coordinates of each scene node are defined as the following linear combination:

$$\text{node} = w_1 p_1 + w_2 p_2 + w_3 p_3. \quad (2)$$

From the viewpoint of computational complexity, the choice of geometric primitive type for mass vectors was a critical architectural decision. The video rendering time is determined primarily by the dynamic animation scenario (e.g., circular camera rotation). Preliminary experiments with full-fledged 3D arrows (`Arrow3D`) showed that the need to render complex spatial surfaces (cones and cylinders) for each of the dozens of vectors caused scene generation time to stretch to several hours, even in low-quality mode (`-q1`). For an authoring system intended for bulk production of educational materials such performance is unacceptable. Consequently, vectors of the local layer are realized as oriented two-dimensional `Arrow` objects placed in three-dimensional space. Their visual attributes are deliberately strengthened (stroke width 4.5, full opacity, zero buffer) so that the didactic information remains salient while generation time is reduced to an acceptable 7–10 minutes.

4.2. Semantic Color Encoding of Local Flux Contribution

To establish robust cognitive associations between the geometric representation and the algebraic concept of integral flux, an algorithm for dynamic semantic coloring of local vectors has been implemented. For each generated grid node of the triangular element, the values of the vector field components (u, v, w) are calculated in the virtual scene coordinate system based on the physical field components (P, Q, R) . Subsequently, the scalar product of the vector field and the normal vector to the plane $n = (A, B, C)$ is computed:

$$d = uA + vB + wC. \quad (3)$$

Depending on the sign of the scalar product (d), the vector is classified by its interaction type with the surface and rendered using a predefined color palette:

- `color_out` (for $d > 10^{-6}$): indicates outward flux, where the vector possesses a positive normal component;
- `color_in` (for $d < -10^{-6}$): indicates inward flux, where the vector possesses a negative normal component;
- `color_tan` (in all other cases): indicates tangential flux, where the vector is purely tangential to the surface.

A small numerical tolerance 10^{-6} is introduced to suppress floating-point artefacts and to ensure reliable identification of the tangential state.

4.3. Spatial Transformations and Perceptual Stabilization of the 3D Scene

Geometric fidelity and perceptual stability under arbitrary camera motion are achieved by a coherent set of spatial transformations.

Dynamic Unit Scaling. This parameter defines the metric of the virtual scene space, representing the number of internal ManimCE engine units per unit interval in the physical coordinate system of the problem. The scaling coefficient k_{unit} is calculated dynamically according to the formula:

$$k_{\text{unit}} = \frac{L_{\text{target}}}{\max(2x_{\text{lim}}, 2y_{\text{lim}}, z_{\text{lim}})}. \quad (4)$$

This maps the physical coordinates of the problem into the relative units of the ManimCE virtual space. The parameters of this model are directly controlled via the graphical interface (Figure 4). The base target size L_{target} corresponds to the *Unit Scale* slider, operating with values converted to a relative scale coefficient ranging from 0.5 to 10.0 (with a default setting of 7.0 pixels, corresponding to an internal slider range of 5 to 100). The spatial boundary parameters x_{lim} , y_{lim} , and z_{lim} are determined by the *X-Axis Limit* (Ox), *Y-Axis Limit* (Oy), and *Z-Axis Limit* (Oz) interface controls, respectively. This integrated approach enables flexible adaptation of the overall visual scale to the geometric proportions of a specific mathematical problem.

Vertical Framing and Interactive Correction (*Z-Shift*). Based on the user-defined limits of the applicate axis (z_{min} and z_{max}), the system automatically calculates the geometric center of the computational domain:

$$z_{\text{mid}} = \frac{z_{\text{min}} + z_{\text{max}}}{2}. \quad (5)$$

This value serves as the primary centering coordinate for the geometry relative to the scene camera. Given that automatic alignment does not always provide an optimal viewpoint due to the variability in the spatial positions of cutting planes, a manual adjustment slider (`slider_z_shift`) is integrated into the interface for precise focusing on the target zone. The final spatial transformation of the axis system within the ManimCE virtual space is performed along the vertical unit vector (`OUT`):

```
axes.shift((-z_mid + z_shift) * OUT)
```

where `z_shift` is the numerical offset value obtained directly from the graphical user interface.

Spatial Orientation of Text Labels. By default, the built-in vectorized text objects in the ManimCE engine are two-dimensional and positioned in the base xOy plane. During 3D visualization and dynamic camera rotation, this configuration leads to significant perspective distortion, potentially rendering the symbols illegible. To maintain readability, an algorithm for spatial reorientation is implemented: coordinate axis labels undergo successive fixed rotations around the local scene axes, ensuring their perpendicularity to the corresponding coordinate axes. This minimizes the geometric text compression effect and guarantees accurate visual interpretation of designations when viewing the 3D scene from various angles.

Stabilization of Interface Information Elements. The two-dimensional counter of the integral flux value (`formula_card`), displayed in the top-right corner of the screen, is isolated from three-dimensional camera rotation using the `add_fixed_in_frame_mobjects()` method. It remains stationary in screen space while the camera orbits the three-dimensional scene, enabling continuous monitoring of the quantitative result. Collectively these geometric and perceptual measures eliminate the characteristic visual artefacts of three-dimensional rendering and guarantee that the educational information remains invariant under arbitrary changes of viewpoint.

5. Implementation and Functional Demonstration

To assess the engineering feasibility of the proposed two-stage visualization algorithm and to determine the optimal spatial discretization parameters, a series of computational time measurements was performed. The computational experiment was carried out on a laptop equipped with a quad-core Intel Core i5-10300H processor (base frequency 2.50 GHz), 16 GB of RAM, a high-performance NVMe SSD, and an NVIDIA GeForce GTX 1650 Ti discrete graphics card. Testing was conducted for the base case of polynomial components of the vector field. Taking into account didactic needs, the highest rendering quality profile (High Quality, 1080p, 60 fps) was considered excessive due to inefficient non-linear growth of time costs. Instead, two modes were analyzed: Low Quality (`-ql`, 480p, 15 fps — for rapid scene generation and verification) and Medium Quality (`-qm`, 720p, 30 fps — as the reference final variant for the didactic package). The influence of background vector density and cross-sectional plane discretization on the generation speed of a 32 s animation clip (full spatial camera orbit) is presented in Table 1. Analysis of the obtained time indicators shows that the Medium Quality mode (720p, 30 fps) offers a practical compromise between visual clarity of geometric elements and generation time for bulk production of didactic packages on the tested hardware. It should be emphasized that this rendering duration represents a one-time, offline background task executed exclusively on the instructor's side during the pre-generation of didactic packages. Since the output distributed to students consists of lightweight CSV data and pre-rendered MP4 video assets, the prospective student-side client tool operates purely as a low-overhead playback and solution-verification environment. Consequently, these rendering time metrics do not impose any computational hardware or real-time GPU performance requirements on student devices.

The visual result of rendering for one of the computational configurations (polynomial field) is shown in Figure 5.

Table 1

Dependence of instructor-side offline batch rendering time on spatial discretization parameters for a 32 s video asset

No.	Background grid density	Plane grid density (vectors per triangle side)	Rendering time, LQ mode (-ql, 480p)	Rendering time, MQ mode (-qm, 720p)
	(vectors per parallelepiped edge)			
1	5	3	3 m 32 s	7 m 05 s
2	5	5	3 m 35 s	7 m 05 s
3	7	5	4 m 10 s	8 m 00 s

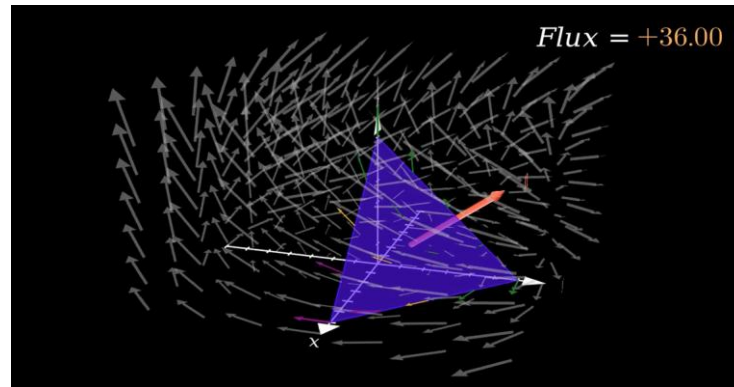


Figure 5: Result of three-dimensional visualization of the interaction between the vector field and the triangular cutting plane in the interface of the developed application.

The obtained graphical output demonstrates a practical solution to the problem of synthesizing a complex three-dimensional scene with a high level of perceptual distinguishability of elements. Spatial coding of vector flows according to their orientation relative to the surface and planar stabilization of interface labels improve the structural readability of the scene regardless of camera viewpoint. The comprehensive methodological effect and the didactic implications of implementing this approach are discussed in detail in the next section.

6. Discussion

The study of the Symbolic-to-Visual Pipeline concept within the framework of automating didactic support for higher mathematics reveals several fundamental interdisciplinary insights at the intersection of computational geometry, symbolic computation, and the cognitive psychology of learning. The primary didactic challenge in teaching vector calculus at technical universities lies in the significant cognitive gap between the symbolic (analytical) representation of differential and integral operators and their spatial geometric meaning. Traditional computer algebra systems (e.g., Maple, MATLAB) are designed primarily for scientific research and data analysis, not for the automated generation of didactic materials. Creating individualized, multi-variant 3D visual assets in such environments requires custom scripting and manual scene configuration, which creates a technical barrier for instructors seeking to scale didactic material production. Analyzing the

conceptual architecture of the proposed Symbolic-to-Visual pipeline for bulk surface flux task production, several key aspects should be highlighted:

1. Mathematical invariance of the computational core. Symbolic parsing based on the SymPy library provides native support for nonlinear vector fields of arbitrary complexity. The computational complexity involved in differentiating or integrating components P , Q , and R is fully mitigated by computer algebra algorithms, shifting the engineering focus from mathematical constraints to topological ones.

Perceptual stability of three-dimensional space. The implementation of spatial orientation mechanisms for text labels and the Z-Shift algorithm addresses a fundamental problem in computer graphics — optical occlusion and perspective distortion of information markers. By ensuring the perpendicularity of glyphs relative to the observation vector and dynamically compensating for empty spaces along the appicate axis, the system preserves the semantic integrity of the scene regardless of camera viewing angles. Color segregation of vectors based on their orientation relative to the cutting plane serves as an additional cognitive filter, preventing visual chaos.

Asynchronous computational decoupling between ecosystem modules. The deliberate separation of the system into an authoring tool (instructor) and an exploration client (student) isolates computationally expensive procedural rendering. By executing the multi-minute 3D animation synthesis on the instructor's hardware prior to the learning process, the student module remains completely decoupled from heavy graphics calculation. Playing pre-rendered MP4 files alongside analytical evaluation ensures smooth, interactive performance even on budget- or legacy-grade student hardware.

Operational alignment relative to Scientific Visualization (SciVis) frameworks. A critical distinction between the proposed software module and established general-purpose SciVis ecosystems (e.g., VTK 15, ParaView 15, and Inviwo 15) lies in the target user persona and the operational workflow. While these SciVis platforms excel at interactive, high-throughput exploratory visual analysis of complex volumetric datasets, they require substantial user training, manual configuration, and are not tailored for the rapid, automated synthesis of individualized educational materials. Similarly, while specialized pedagogical platforms like FlowVisual 15 provide interactive demonstrations of vector flows, they primarily focus on real-time student exploration rather than bulk authoring workflows for instructors. The proposed conceptual architectural model bridges a specific integration gap: the simultaneous satisfaction of mathematical invariance of vector fields, perceptual stability of the 3D scene, zero-code interaction and deterministic bulk production of synchronized packages — constraints that are not jointly addressed by either general-purpose scientific visualization platforms or existing Manim-based educational tools.

Integration domain scope and pedagogical alignment. At the current stage, the system focuses on the foundational class of integration surfaces — planar regions intersecting the three coordinate axes. This choice directly aligns with standard vector calculus curricula, where instruction in evaluation techniques for surface integrals of the second kind traditionally begins with bounded planar domains before progressing to complex curved surfaces.

Thus, the proposed toolkit does not merely automate the routine work of an instructor in generating didactic materials; it establishes a methodological foundation for transitioning from passive visualization to an interactive modeling environment. Transforming abstract symbolic notation into a dynamic spatial image allows for overcoming perceptual barriers in the understanding of higher mathematics, ensuring the formation of robust mental models in future engineers.

7. Conclusions

Within the framework of this study, a significant scientific and practical problem has been addressed: the formulation and experimental validation of a conceptual architectural model of an instructor-oriented Symbolic-to-Visual pipeline for the bulk production of individualized surface flux learning materials.

The results of the work allow for the following conclusions to be drawn:

1. The proposed model establishes structured data serialization protocols between the symbolic engine and the procedural renderer, enabling a deterministic framework that transforms instructor-defined mathematical models into synchronized multi-variant textual-visual didactic packages while preserving mathematical invariance and perceptual stability of the 3D scene.
2. Systemic limitations of procedural rendering of 3D scenes in geometric animation engines have been identified and overcome. Through the development of fixed matrix rotation algorithms for 2D text objects and interactive vertical shift compensation (Z-Shift), a perceptual invariance effect has been achieved, ensuring total readability and informativeness of graphical elements regardless of virtual camera viewing angles.
3. The computational and architectural viability of the proposed Symbolic-to-Visual pipeline has been experimentally confirmed on consumer-grade hardware (CPU Intel i5-10300H / GPU GTX 1650 Ti). It was determined that the Medium Quality rendering profile (720p, 30 fps) serves as a didactic and computational compromise, enabling mass asynchronous generation of unique examination or practical task variants without the need for server capacity or cloud render farms.

A promising direction for further research in this context is the primary development of a paired autonomous training complex for students, built on the principles of interactive scaffolding. The application will implement a step-by-step analytical control algorithm with differentiated diagnostics of intermediate errors. Within this framework, dynamic 3D visualization serves not merely as a passive illustration, but as a cognitive reward that provides deeper spatial insights into the successfully solved problem. Further scaling of the ecosystem is planned in the format of linked software pairs (instructor's generative module and student's training complex) along two strategic avenues: extending the class of supported integration surfaces to general curvilinear surfaces, and developing consistent domain-specific modules for modeling specialized vector fields within professional engineering disciplines.

Declaration on Generative AI

During the preparation of this manuscript, the authors used LLM Gemini as an AI co-pilot and technical assistant in order for:

- Code Implementation and syntax assistance. The AI was used to generate specific Python code snippets, verify syntax, and troubleshoot component integration across the PyQt6, SymPy, and ManimCE libraries. The overall software design, system architecture, and scientific contributions were developed entirely by the authors.
- Academic Refinement and Narrative Editing. The tool was utilized to refine manuscript structure, stylistic consistency, and academic tone. This included polishing the scientific narrative of the discussion and conclusion sections, eliminating repetitive phrasings, and ensuring the precise use of technical terminology in English.

After all interactions with the AI tool, the authors performed a rigorous manual review, cross-verification, and final editing of all generated content. The authors bear full responsibility for the research methodology, the correctness and integrity of the software implementation, and the final content of this publication.

References

- [1] J. Schoenherr, A. R. Strohmaier, S. Schukajlow, Learning with visualizations helps: A meta-analysis of visualization interventions in mathematics education, *Educational Research Review* 45 (10) (2024) 100639. doi:10.1016/j.edurev.2024.100639.
- [2] M. O. Fokuo, N. Opuku-Mensah, R. Asamoah, J. Nyarko, K. D. Agyeman, et al., The use of visualization tools in teaching mathematics in college of education: A systematic review, *Online Journal of Mathematics, Science and Technology Education* 4 (2023) 65–75.
- [3] R. Lowe, J.-M. Boucheix, Towards more Valid Assessment of Learning from Animations, in: L. Unsworth (Ed.), *Learning from Animations in Science Education*, volume 25 of *Innovations in Science Education and Technology*, Springer, Cham, 2020, pp. 301–322. doi:10.1007/978-3-030-56047-8_13.
- [4] K. H. Cheong, J. S. Chen, K. Kang, D. J. Yeo, Supporting students' visualization of multivariable calculus partial derivatives via virtual reality, *Mathematics* 11 (4) (2023) 831. doi:10.3390/math11040831.
- [5] Pinter, M. F. H. Siddiqui, Enhancing calculus learning through interactive VR and AR technologies, *Multimodal Technologies and Interaction* 8 (3) (2024) 19. doi:10.3390/mti8030019.
- [6] 3Blue1Brown, Site of 3Blue1Brown: Explaining math and physics with animation. URL: <https://www.3blue1brown.com/>.
- [7] J. Tian, Manim-Based Animations in STEM Education: A Literature Review, *Computer Applications in Engineering Education* 34 (2026) e70230. doi:10.1002/cae.70230.
- [8] C. Zhang, Manim for STEM education: Visualizing complex problems through animation, *arXiv preprint arXiv:2510.01187* (2025). doi:10.48550/arXiv.2510.01187.
- [9] Y. Niu, L. Wang, Y. Wang, P. Wang, Study on improving the teaching effect of mathematical methods for physics using Manim animation technology, *Journal of Contemporary Educational Research* 9 (10) (2025) 215–222. doi:10.26689/jcer.v9i10.12674.
- [10] P. Akhilesh, K. Amal Krishna, S. K. Bharadwaj, D. Subham, M. Belwal, A Visual Approach to Understand Parsing Algorithms through Python and Manim, in: *2024 15th International Conference on Computing Communication and Networking Technologies (ICCCNT)*, IEEE, Kamand, India, 2024, pp. 1–7. doi:10.1109/ICCCNT61001.2024.10725029.
- [11] M. Marković, I. Kaštelan, Demonstrating the Potential of Visualization in Education with the Manim Python Library: Examples from Algorithms and Data Structures, in: *2024 47th MIPRO ICT and Electronics Convention (MIPRO)*, IEEE, Opatija, Croatia, 2024, pp. 625–629. doi:10.1109/MIPRO60963.2024.10569661.
- [12] P. Samarth, V. Jain, S. Golugula, M. S. Sathvik, Manimator: Transforming Research Papers into Visual Explanations, *arXiv preprint arXiv:2507.14306* (2025). doi:10.48550/arXiv.2507.14306.

- [13] F. Schneider, P. Libbrecht, V. Mas, Creating animations for maths teaching: A praxis-oriented discussion paper about the use chatbots for the creation of mathematical animations with Manim, IU Discussion Papers (2025). doi:10.56250/4092.
- [14] Y. Noviello, A. Birillo, G. Migut, ANVIL: Analogies and Videos for Lecturers, in: E. G. Blanchard, G. Chen, M. Chi, S. Isotani (Eds.), Artificial Intelligence in Education. AIED 2026, volume 16581 of Lecture Notes in Computer Science, Springer, Cham, 2027, pp. 340–354. doi:10.1007/978-3-032-29744-0_23.
- [15] Manim Community Developers, Mobjects not remaining fixed in frame during camera rotation, Manim issue tracker. URL: <https://github.com/ManimCommunity/manim/issues>.
- [16] W. Schroeder, K. Martin, B. Lorensen, The Visualization Toolkit (VTK): An Object-Oriented Approach to 3D Graphics, 4th Edition, Kitware, 2006. URL: <https://vtk.org/>.
- [17] J. Ahrens, B. Geveci, C. Law, ParaView: An End-User Architecture for Large Data Visualization, in: C. D. Hansen, C. R. Johnson (Eds.), Visualization Handbook, Butterworth-Heinemann, Burlington, 2005, pp. 717–731. doi:10.1016/B978-012387582-2/50038-1.
- [18] D. Jönsson, P. Steneteg, E. Sundén, R. Englund, S. Kottraval, et al., Inviwo — A Visualization System with Usage Abstraction Levels, IEEE Transactions on Visualization and Computer Graphics 26 (12) (2020) 3441–3454. doi:10.1109/TVCG.2019.2920639.
- [19] C. Wang, VisVisual: A Toolkit for Teaching and Learning Data Visualization, IEEE Computer Graphics and Applications 42 (4) (2022) 101–108. doi:10.1109/MCG.2022.3176199.
- [20] H. Y. Tuluchenko, Analysis of methodical support for teaching vector analysis and approaches to its improvement, in: H. D. Berehova (Ed.), Theory and methods of teaching in technical higher education institutions, Knyzhkove vydavnytstvo FOP Vyshemyrskyi V.S., Kherson, 2025, pp. 253–277.
- [21] H. Y. Tuluchenko, Interactive visualization of vector calculus: Development of a gamified educational application using the Manim library, in: Proceedings of VII International Scientific and Practical Conference "Innovative Technologies in the Development of Modern Society", Lviv, Ukraine, 2025, pp. 50–54.
- [22] H. Y. Tuluchenko, T. P. Bilousova, Verification potential of custom-built software modules in the mathematical training of engineers amid digitalization, Modern Information Technologies and Innovation Methodologies of Education in Professional Training: Methodology, Theory, Experience, Problems 79 (2026) 144–155. doi:10.31652/2412-1142-2026-79-144-156.