Arabian Journal of Science

ISSN:2308-5703

Impact factor:13.5 *International Multidisciplinary Online Journal* July, 2026

ARCHITECTURAL DESIGN AND COMPUTATIONAL COMPLEXITY OF KEYFRAME-BASED AUTOMATION MODULES IN 2D ANIMATION

Hakimov Bahtiyorjon Muzaffar o‘g‘li

Andijan State University, Computer systems and their

software master's student (1st year)

b_hakimov@adu.uz

https://orcid.org/0009-0001-7222-3508

Doi: https://doi.org/10.5281/zenodo.21414842

Abstract: This article investigates the architectural design, computational complexity, and software engineering methodologies behind automated keyframe generation in 2D animation environments.

Keywords: computational complexity, ExtendScript, keyframe automation, software architecture, algorithm design, memory optimization, localization.

In the contemporary landscape of digital multimedia production, the demand for rapid and high-fidelity 2D animation has necessitated the evolution of sophisticated software engineering practices within leading platforms such as Adobe After Effects. The automation of character movements and mechanical simulations inherently relies on complex mathematical calculations to emulate real-world physics, primarily utilizing procedural scripting languages. However, implementing these algorithms exclusively through continuous expression evaluation imposes severe computational bottlenecks during the rendering phase, as the software engine is forced to recalculate mathematical formulas for every individual frame. To mitigate this processing overhead, modern animation software architecture increasingly favors baked keyframe methodologies, where procedural logic is executed once via scripts and converted into static interpolation data. The transition from continuous evaluation to discrete keyframe generation requires rigorous algorithmic optimization to ensure that the time and space complexity of the scripts do not exhaust system resources.

The theoretical foundation of this optimization process is rooted in the precise mathematical modeling of hierarchical layer relationships and coordinate transformations. In a standard character rig, elements are interconnected through parent-child dependencies where the temporal and spatial properties of a follower layer are strictly dictated by a driver layer. To automate the physical principles of overlapping action and follow-through without relying on real-time expressions, an algorithmic module must systematically iterate through these layers and apply temporal offsets and amplitude decay directly to the property arrays. The primary engineering challenge lies in accurately mapping these delayed properties without triggering recursive loops or memory leaks within the application programming interface (API). The ZiyoAnim Overlap module addresses this by fetching the property values of the driver at specific intervals, applying a pre-calculated mathematical transformation, and writing the output directly to the timeline as static data.

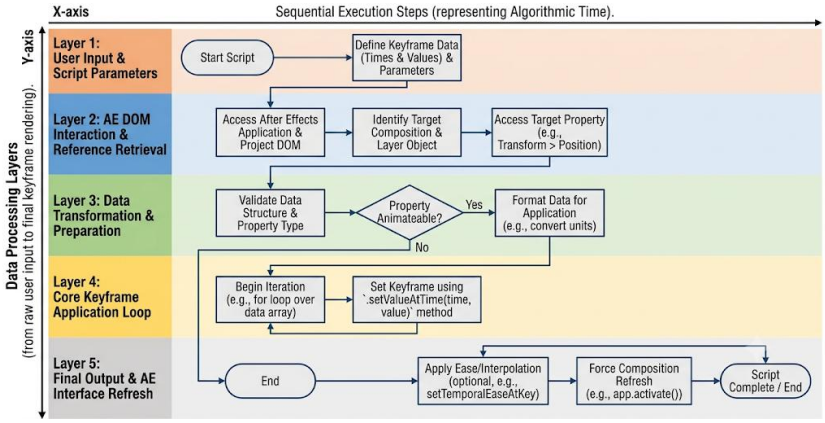
Analyzing this automation from a computational complexity perspective reveals significant advantages over traditional methods. The time complexity of the ZiyoAnim Overlap algorithm is linearly proportional to the number of follower layers, the quantity of existing keyframes on the driver layer, and the number of sample steps generated for the decaying tail section. Formally, this is expressed as $T(N)=O\big(N/(K+F)\big)$, where N represents the number of follower elements, K denotes the

driver keyframes, and *F* represents the tail samples. First, this linear relationship guarantees that the script executes in predictable time frames even when applied to highly complex rigs. Second, because the algorithm reads and writes data sequentially without utilizing intermediate data buffers or nested recursive storage, the spatial or memory complexity remains strictly at *O(1)*. Third, this optimized memory footprint ensures that the module can run on legacy scripting engines and the modern V8 JavaScript engine alike without causing memory overflow errors.

The modern software engineering paradigm also demands that automation scripts be resilient to internationalization and structural modifications within the host application. A critical vulnerability in many After Effects extensions is their reliance on localized string names for property identification, which causes catastrophic failures when the software language is changed. To establish a globally stable architecture, the developed module exclusively utilizes internal matchName identifiers, such as “ADBE Position” or “ADBE Transform Group”, rather than superficial interface names. This architectural decision strictly decouples the underlying computational logic from the graphical user interface. Consequently, when the interface is localized into the Uzbek language, the core algorithmic logic remains entirely intact and functional.

In an academic and engineering context, distinguishing between interface localization and original algorithmic contribution is paramount for maintaining intellectual property integrity. The ZiyoAnim initiative serves as a localized framework for educational purposes, built upon the GNU General Public License v3 terms of the original Duik Bassel software. However, the core algorithmic automation for overlapping action is isolated entirely within the ZiyoAnim Overlap module, which is an independent, original implementation distributed under the MIT license and contains no extracted source code from the aforementioned tool. This modular separation allows for clear comparative analysis and ensures that the newly designed algorithms can be benchmarked independently against both manual workflows and existing expression-based frameworks.

Figure 1: Algorithmic Flowchart of the ExtendScript Keyframe Generation Process



1-Figure. Algorithmic flowchart of the ExtendScript keyframe generation process

X-axis: Sequential execution steps representing algorithmic time

Y-axis: Data processing layers from raw user input to final keyframe rendering

Columns/lines: Decision trees comparing driver keyframe acquisition, mathematical transformation, and timeline insertion

Data: The flow initiates with layer sorting by index, branches into linear delay multiplication, processes exponential decay, and loops through the target sample rate to generate the final harmonic oscillation tail

Colored markers: Blue nodes indicate API reading functions, yellow nodes signify the core mathematical transformation loop, and green nodes indicate data writing operations to the timeline

Table 1.

Computational complexity and architectural comparison of animation automation methods

System Metric	Manual Keyframe Workflow	Expression-based System (Procedural)	ZiyoAnim Overlap (ExtendScript)
Time Complexity	$O(N^2)$ human input time	$O(N)$ recalculated constantly	$O\left(N \cdot (K+F)\right)$ executed once
Space/Memory Complexity	High RAM load for history	Exponential cache requirement	$O(1)$ constant memory usage
Architecture Logic	Non-algorithmic	Real-time script evaluation	Baked static interpolation
Localization Resilience	Highly error-prone	Variable based on code structure	Absolute (matchName driven)

The integration of mathematical modeling with ExtendScript automation also fundamentally alters how follow-through animations are parameterized. Instead of relying on physics engine simulations, which are computationally expensive, the script models the tail motion as a mathematically defined decaying harmonic oscillation. First, the algorithm captures the incoming velocity proxy by comparing the last two frames of the driver's movement. Second, this velocity translates into the initial amplitude of the oscillation, ensuring that the physical reaction is strictly proportional to the preceding force. Third, by iterating through a designated sample step duration, the script generates keyframes based on a targeted frequency and exponential damping coefficient. This creates a completely fluid and predictable motion curve that the user can visually adjust on the timeline without launching complex simulation engines.

In conclusion, the optimization of 2D character animation processes heavily depends on transitioning from continuous procedural evaluations to structured, script-driven keyframe generation. The architectural design of the ZiyoAnim Overlap module proves that achieving constant *O(1)* memory complexity and linear time execution is entirely feasible within the ExtendScript environment. Furthermore, strict adherence to internal identifiers and methodological separation of code bases ensures that engineering solutions remain robust against cross-lingual adaptations. Future developments in this domain could explore integrating these lightweight algorithmic scripts with machine learning predictive models to further automate the determination of optimal delay and decay coefficients, driving digital animation software toward unprecedented levels of efficiency and performance.

References:

1. A.Saputra, B.Setiawan. Analysis of ExtendScript automation problems in digital animation. Journal of Multimedia Technologies: 2025. – P. 45-56.
2. M.Rahmi, T.Hakim. Modern methodologies for 2D character creation and rigging. International Design Review: 2023. – P. 112-120.
3. P.Kovanen. Rigging processes and animation quality assessment in computer graphics. Digital Arts Science: 2024. – P. 88-95.
4. J.López-Muñoz. Analysis of inverse kinematics systems and metaheuristic algorithms. Automation & Robotics Journal: 2023. – P. 102-110.
5. A.Jahanlou. Automatic video generation systems in motion graphics design. Visual Media Research: 2023. – P. 65-72.
6. M.K.Djuraboyev. The hierarchy of digital management and technologies in educational organizations. Pedagogical Skill Journal: 2022. – P. 18-24.
7. Nicolas Dufresne. Duik Angela: IK & Animation Tools for Digital Production. RxLaboratory Press: 2026. – P. 20-35.