MSc Computer Animation & Visual Effects

Master's Project

Student:
Joshua Alexander Medina Gracia
s5820775

Supervisors:
Jon Macey
Jian Chang

MSc Project Dissertation

Title:
AOVGUARD

University:
Bournemouth University

National Centre for Computer Animation

August 2026

AOVGUARD: A Modular EXR and AOV Validation Framework

for VFX Lighting and Rendering Pipelines

A production-oriented study of EXR structure inspection, contextual validation, shared interfaces and repeatable software-quality evidence.

Contents

Abstract

Rendered OpenEXR files are central to contemporary visual-effects workflows because they can preserve high-dynamic-range image data, metadata and many Arbitrary Output Variables (AOVs) in a single container. The same flexibility also makes technical inspection difficult: a file may be readable while still containing missing, empty, inconsistent or non-finite data that causes avoidable downstream work. This dissertation presents AOVGuard 2.0, an applied software-engineering project that develops a Python prototype into a modular EXR and AOV validation tool for lighting, rendering, compositing and pipeline Technical Director workflows.

The project replaces manual "Simple" and "Multilayer" modes with automatic structure inspection. It uses OpenEXR as the selected production reader, represents colour internally as RGB, centralises configurable Rec.709, Rec.601 and custom weighted brightness calculations, discovers input files once, processes each supported frame once at the application boundary, and executes AOV-aware validation rules loaded from TOML or JSON. A common `AnalysisReport` model is consumed by both a command-line interface and a PySide6 graphical interface, and can be exported as strict JSON or self-contained HTML.

Evaluation combines automated correctness tests, branch-aware coverage, authorised real EXR validation and a controlled architectural benchmark. The final verified test run completed 229 tests with 95.24% combined statement and branch coverage. Additional workflow validation processed a numbered Maya/Arnold sequence of 61 EXRs with no failed reads, while identifying 51 channels, 18 AOV descriptors and seven contextual warnings. A separate 12-frame, five-AOV benchmark with 30 repetitions per strategy reduced application-level reader calls from 60 to 12 and reduced median elapsed time from 0.4387 seconds to 0.1181 seconds while preserving the output checksum. These results support the chosen architecture under the recorded conditions, but do not establish universal performance or production readiness. Deep and multipart analysis, renderer-specific semantics, colour-space-aware brightness interpretation and large-scale native-memory profiling remain limitations. The project contribution is therefore a focused validation workflow with repeatable evidence and support for independent reproduction, integrating established EXR, image-statistics and software-quality concepts rather than replacing mature image-processing tools or artistic judgement.

Keywords: OpenEXR, AOV, visual effects, render validation, pipeline TD, lighting, quality control, PySide6, software testing.

GITHUB: https://github.com/Joshmedinag/NCCA-mastersproject.git

1. Introduction

1.1 Production context

Visual-effects production depends on the reliable movement of image data between rendering, lighting, compositing and review. A rendered frame is rarely only a final RGB image. Modern renderers can produce a beauty image together with diffuse, specular, emission, albedo, depth, position, normal and mask information. These outputs are commonly represented as AOVs and may be merged into a multilayer OpenEXR file. This arrangement gives compositors flexibility to rebalance components and diagnose problems without requesting a complete re-render, but it also increases the amount and structural complexity of data that must be checked.

OpenEXR is well suited to this workflow because it supports floating-point pixels, arbitrary named channels, metadata, multiple parts and deep data (Academy Software Foundation, n.d.-c). A file can therefore be syntactically valid while still being unsuitable for its intended handoff. An expected pass may be absent; an emission pass may be entirely black; one frame may have different dimensions or channels from the remainder of a sequence; or a numerical failure may introduce NaN or infinity values. Human inspection can identify some of these conditions, but manual checking becomes repetitive and error-prone when a delivery contains many frames and AOVs.

AOVGuard began with a deliberately narrow practical question: how could a Lighting or Pipeline TD determine quickly whether a rendered pass contained a meaningful pixel contribution before it moved further through a pipeline? The first answer was a small Python prototype that distinguished active, nearly empty and empty render passes. It demonstrated the usefulness of a dedicated validation step and provided both command-line and graphical access, allowing the same idea to be explored from a scripting and artist-facing perspective.

That early success also exposed architectural limitations. Users selected "Simple" or "Multilayer" mode manually, the two paths used different image libraries and channel conventions, brightness calculations were duplicated, frame discovery was not a single source of truth, and the multilayer flow could open each frame once for every AOV. These limitations made additional features increasingly risky because the same concept could behave differently in the CLI, GUI or analyser. The development journey therefore became part of the research method: instead of adding checks directly to the prototype, the project examined why the prototype was difficult to extend and used those findings to shape AOVGuard 2.0. Section 3.4 records that evolution and its principal decision points.

1.2 Problem statement

The central problem addressed by this project is not whether OpenEXR can be read. Mature tools already inspect and process EXR files. The problem is how to combine structure inspection, AOV interpretation, pixel metrics, sequence checks, configurable validation, reporting and artist-accessible presentation into one coherent workflow. The tool must remain technically cautious

because unusual values are not automatically wrong in scene-linear HDR imagery. Values greater than one can be valid, negative values may be expected in some technical or colour-processing contexts, and a constant mask may be deliberate. Automated validation should therefore provide evidence and context rather than assert artistic correctness.

The project also addresses a software-engineering problem. A production-oriented validator needs repeatable behaviour across interfaces, a clear boundary around image IO, structured results, isolated rule failures and tests that target edge cases rather than only successful examples. ISO/IEC 25010:2023 identifies characteristics including functional suitability, performance efficiency, interaction capability, reliability, maintainability and flexibility as dimensions of software product quality (International Organization for Standardization & International Electrotechnical Commission, 2023). AOVGuard is not claimed to conform formally to that standard, but those characteristics provide a useful vocabulary for evaluation.

1.3 Aim, objectives and research questions

The aim is to design, implement and evaluate a modular OpenEXR/AOV validation workflow suitable for a focused MSc project and relevant to Lighting TD and Pipeline TD practice.

The objectives are:

1. to inspect EXR structure automatically instead of requiring a manual simple/multilayer decision;
2. to establish a canonical internal RGB convention and a centralised brightness metric;
3. to process discovered frames consistently and avoid repeated application-level reads per AOV;
4. to separate validation rules from the main processing loop and load their configuration from files;
5. to expose the same backend through CLI and GUI workflows;
6. to produce deterministic, machine-readable reports; and
7. to evaluate correctness, coverage, performance, usability risks and limitations.

These objectives lead to four research questions:

RQ1: Can a structure-first model remove the user's manual simple/multilayer choice while describing the channels and AOVs required by the MVP?

RQ2: Can a shared, AOV-aware rule system provide consistent validation behaviour across CLI, GUI and reports?

RQ3: Does frame-first processing reduce application-level reader calls and elapsed time relative to the original AOV-first organisation under a controlled benchmark?

RQ4: What evidence is sufficient to support the tool's correctness and maintainability without overstating code coverage, synthetic fixtures or small-scale performance measurements?

1.4 Scope and contribution

The implemented scope includes automatic inspection, named-channel grouping, broad AOV categorisation, RGB normalisation, configurable brightness weights, NaN/Inf detection, empty and near-empty checks, channel and AOV consistency, sequence discovery, frame-first analysis, JSON/HTML reports, baseline comparison, a CLI and a responsive GUI. Deep and multipart EXRs are detected but intentionally rejected by the MVP analysis path. External Python plugins, complete renderer presets, pass-specific semantic validation and direct DCC or render-farm integration remain future work.

The contribution is best described as applied VFX software engineering. AOVGuard does not invent HDR imaging, AOVs, OpenEXR, image statistics or automated QC. It investigates how these established ideas can be integrated into a specialised validation workflow with explicit data models, measurable processing behaviour and repeatable evidence. This position avoids claiming novelty where mature tools already exist, while identifying the project's practical contribution: translating low-level EXR information into AOV-aware findings usable from both pipeline scripts and an artist-facing interface. Figure 1 contrasts the manually selected workflows of the original prototype with the shared automatic workflow developed for AOVGuard 2.0.

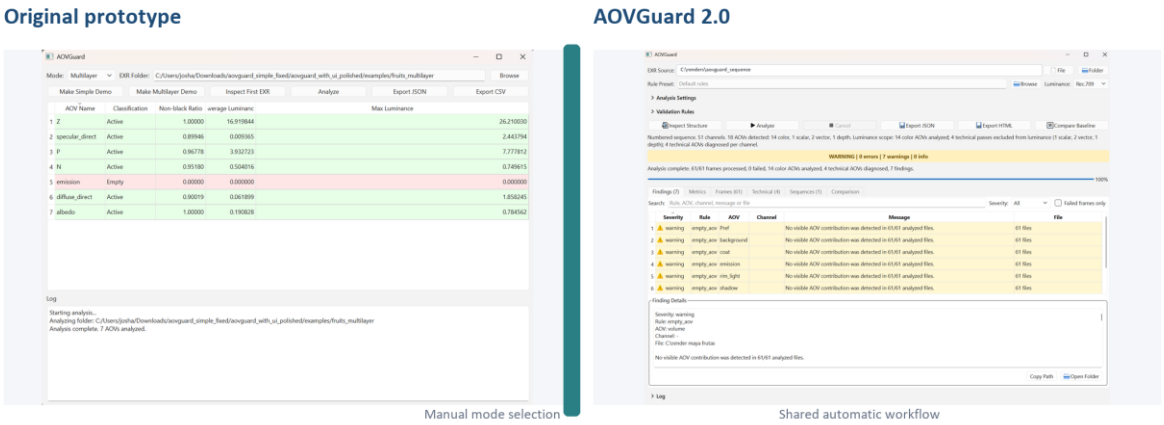


Figure 1. Evolution from the original manually selected analysis modes (left) to the current shared automatic workflow (right). The historical interface is retained only as development evidence.

2. Related Work and Technical Background

2.1 OpenEXR as a structured image container

OpenEXR was designed for high-quality image processing and storage. Its feature set includes 16-bit and 32-bit floating-point channels, arbitrary attributes, tiled or scanline storage, multiview images and deep data (Academy Software Foundation, n.d.-c). An EXR is consequently better understood as a structured image container than as a conventional RGB bitmap. Channels may be named `R`, `G`, `B` and `A`, but they may also use layered names such as `diffuse_direct.R`, `N.X` or `crypto_object00`.

The terms simple, multichannel, multilayer, multipart and deep describe different properties and should not be treated as a single binary classification. A single-part EXR can contain only RGB, RGB plus additional channels, or many channels organised by dotted layer names. A multipart EXR stores more than one independently described image part. Deep files store a variable number of samples per pixel and require different compositing semantics. The OpenEXR Python API represents files as one or more `Part` objects and exposes header metadata and NumPy pixel arrays (Academy Software Foundation, n.d.-a). AOVGuard therefore records dimensions, channel names, AOV descriptors, part count and deep status rather than returning only "simple" or "multilayer".

This distinction is important for reliable failure handling. The MVP does not silently flatten deep or multipart inputs. It inspects their structure and returns a clear unsupported reason. This is a narrower claim than complete OpenEXR support, but it is preferable to producing plausible-looking metrics from a structure that the implementation does not understand.

2.2 AOVs and compositing workflows

AOVs separate components of a rendered image so that downstream departments can inspect or recombine them. Autodesk's Arnold documentation describes AOV output and merged EXR workflows in which several AOV channels are written into one file (Autodesk, 2024a, 2024b). In practice, naming and channel organisation differ between renderers, studios and shows. A beauty output may appear as root `R/G/B/A`, while a normal pass may use `N.R/N.G/N.B`, `N.X/N.Y/N.Z`, or a show-specific name.

The complexity is not merely theoretical. Alway et al. (2018) describe managed AOV manipulation in a production compositing context, demonstrating that relationships between passes can become difficult to maintain. Brinkmann (2008) and Wright (2024) similarly place render passes, mattes and auxiliary data within the wider craft of digital compositing. AOVGuard addresses a limited point in that workflow: technical validation before render data move downstream. It cannot determine whether the lighting is appealing or whether a compositor intends to use a pass, but it can identify measurable conditions that deserve review.

The application groups channels into approximate categories: colour, scalar, vector, mask, depth and unknown. Colour AOVs can receive RGB brightness metrics; vectors and depth receive per-channel diagnostics; unknown structures are retained rather than forced into a colour interpretation. This

model is intentionally heuristic. Names such as `N`, `P` and `Z` provide strong conventions, but no universal naming scheme guarantees the semantic meaning of every custom pass. Presets and future renderer-aware rules are therefore necessary extensions.

2.3 HDR, scene-linear data and weighted brightness

Debevec and Malik's (1997) HDR work established methods for representing radiance beyond the limited range of conventional display images. Production rendering extends this principle through scene-referred, floating-point data. ACES2065-1 and ACEScg are photometrically linear encodings used for interchange and computer-graphics work respectively (Academy of Motion Picture Arts and Sciences, 2025a, 2025b). ACEScg uses a linear transfer function; values above 1.0 are expected in scene-referred work, and negative components can validly represent colours outside the AP1 gamut. Such values are therefore not automatically clipped or erroneous, and numerical interpretation should not assume an eight-bit display range.

AOVGuard calculates a brightness proxy for colour AOVs using weighted RGB sums. The implemented presets are:

Rec.709: $Y = 0.2126R + 0.7152G + 0.0722B$

Rec.601: $Y = 0.299R + 0.587G + 0.114B$

The coefficients are drawn from recognised television recommendations (International Telecommunication Union Radiocommunication Sector, 2011, 2015), but the term "luminance" must be used carefully. The correct relationship depends on whether RGB values are linear or non-linear and on the primaries of the source colour space. A merged Arnold EXR may contain ACEScg data, for example, while Rec.709 coefficients assume a different set of primaries. AOVGuard's configurable weights make the calculation deterministic and adaptable, but the current tool does not inspect colour-space metadata and cannot guarantee a physically meaningful luminance measurement. In this dissertation, the result is treated as a configurable relative brightness metric unless the input encoding is independently known.

The original prototype's more immediate problem was channel order. OpenCV decodes colour images in BGR order by default, whereas named OpenEXR channels can be assembled explicitly as RGB (OpenCV contributors, 2025). Separate formulas compensated for this in places, so the issue was partly conceptual rather than always a numerical error. Maintaining two representations nevertheless made future changes unsafe. AOVGuard 2.0 reconstructs colour from named channel roles and normalises it to RGB at the reader boundary. All downstream calculations then consume the same channel order.

2.4 Non-finite and contextual values

IEEE 754 defines NaN and positive and negative infinity as floating-point values with special semantics (IEEE, 2019). They can propagate through arithmetic and interfere with averages, comparisons or serialization. Detecting them is an objective validation task. AOVGuard counts each

non-finite type and emits structured findings before aggregate classifications can hide the problem. Strict JSON also requires special handling because NaN and Infinity are not permitted JSON numbers under RFC 8259 (Bray, 2017). The report replaces non-finite metric values with `null` while preserving explicit counts and findings.

Negative finite values are more contextual. They may indicate an unexpected render or compositing operation, but can also be legitimate in vector passes or wide-gamut processing. The rule therefore supports AOV categories and configurable severity rather than defining every negative value as a failure. The same principle applies to constant channels, empty masks and high HDR values: technical evidence is useful, but severity depends on pass meaning and production policy.

2.5 Existing inspection and quality-control tools

OpenEXR distributes utilities such as `exrcheck`, `exrheader`, `exrinfo` and `exrmetrics`; OpenImageIO provides mature tools such as `oiiotool` and `idiff` (Academy Software Foundation, n.d.-b; OpenImageIO contributors, n.d.-b, n.d.-c). These tools offer low-level verification, metadata inspection, statistics, conversion and comparison. AOVGuard does not attempt to replace them.

The distinction is the workflow layer. AOVGuard combines structure discovery, heuristic AOV grouping, rules that understand broad AOV categories, sequence checks, structured findings, common CLI/GUI behaviour and reports intended for handoff review. Table 1 summarises the positioning.

Table 1. Positioning of AOVGuard relative to existing inspection and production tools.

System	Primary strength	Relationship to AOVGuard
OpenEXR command-line tools	Format-level inspection, validation and metadata utilities	Lower-level foundation and diagnostic reference
OpenImageIO / oiiotool / idiff	Broad image IO, statistics, processing, caching and comparison	More mature general image toolkit; evaluated as a reader candidate
Renderer/DCC AOV tools	Authoring and export of renderer-specific passes	Upstream source of AOV structures and naming conventions
AOVGuard 2.0	AOV-aware validation workflow, sequence context, findings and shared CLI/GUI reporting	Focused applied integration rather than a replacement image library

Production QC research provides a further point of comparison. Cohen Bengio et al. (2018) describe an application-agnostic sanity-check framework, analytics and review tools used to improve asset quality at ILM. Kurtz et al. (2025) report integrated QC initiatives at DNEG Animation and connect them to production efficiency and tool reliability. Pearsall et al. (2026) discuss movement beyond isolated hardcoded checks in a layout-to-animation handoff context. These systems address broader

pipelines and operate at different scales, but they support the premise that configurable automated checks can prevent downstream cost when used alongside human review.

Haddon et al. (2016) present Gaffer as a reusable computation and Qt-based application framework used in VFX production. AOVGuard is far smaller, but the comparison reinforces the value of separating reusable computation from interface code. The shared backend is therefore not only a code-cleanliness exercise; it is a prerequisite for ensuring that an artist in the GUI and a pipeline script in the CLI receive equivalent analysis.

2.6 Testing and quality evidence

Automated tests are central to the project because channel ordering, missing data, non-finite values and sequence naming are easy to mishandle without visible application failure. Ammann and Offutt (2017) describe criteria-based testing as a systematic approach to selecting tests, while Inozemtseva and Holmes (2014) demonstrate that high code coverage is not strongly correlated with fault-detection effectiveness when suite size is controlled. Coverage is therefore used here to locate unexecuted logic, not as proof that the application is defect-free.

2.7 Synthesis and research gap

The reviewed work separates into three complementary groups. OpenEXR, OpenImageIO and renderer documentation establish format capabilities and authoritative API behaviour. Compositing literature explains why passes and auxiliary data matter in practice. Published studio presentations demonstrate that automated quality control can reduce downstream disruption, while software-testing research explains why such claims require evidence beyond a headline coverage percentage. No single group, however, supplies the complete workflow addressed here.

The gap is not the absence of an EXR decoder or a general image-processing utility. Those areas are already mature. The narrower opportunity is a transparent, inspectable workflow that converts named EXR structure into approximate AOV meaning, applies contextual rules, preserves evidence in a common report, and behaves consistently in artist-facing and scripted interfaces. Large studio systems demonstrate the production value of this idea but are broader, environment-specific and not generally available for examination; low-level utilities expose the data but do not provide AOV-aware delivery policy.

The source base also has limitations. Vendor and project documentation is authoritative for supported features but is not independent evidence of workflow effectiveness. Conference production reports provide valuable practitioner experience but usually disclose less implementation and evaluation detail than archival empirical studies. This dissertation therefore uses these sources to motivate requirements rather than to claim that AOVGuard reproduces a studio platform.

Four design implications follow from the synthesis: inspect structure before selecting analysis behaviour; reconstruct colour from named roles and maintain one RGB convention; separate contextual rules from interfaces; and evaluate the artefact through triangulated evidence comprising

deterministic tests, authorised real files, interface equivalence and a controlled benchmark. AOVGuard is consequently evaluated as a specialised MSc prototype with repeatable evidence and explicit limits, not as a complete production QC system.

3. Requirements and Methodology

3.1 Characterisation of the original prototype

Development began with inspection rather than immediate replacement. Characterisation tests were used to document the original behaviour of `simple.py`, `multilayer.py`, CLI configuration and file discovery. The audit confirmed a split architecture: OpenCV read simple images as BGR arrays, while OpenEXR named channels were assembled in RGB order (OpenCV contributors, 2025). Brightness logic appeared in both analysers. The GUI and CLI selected the analyser explicitly, and the multilayer organisation iterated by AOV across frames.

The initial package coverage was estimated at approximately 24% during the early audit. That historical figure was captured before the current coverage configuration and is not treated as directly comparable to the final measurement. Its value was diagnostic: central modules such as the multilayer analyser, CLI and GUI had little or no automated execution. The response was not to chase a percentage in isolation, but to define edge cases that represented the main technical risks.

3.2 Requirements and scope control

Requirements were prioritised using a MoSCoW approach. Must-have work included automatic inspection, RGB consistency, centralised brightness, NaN/Inf detection, one authoritative discovery result, frame-first processing, configurable rules, a shared API, a CLI, JSON reporting, basic GUI migration and meaningful tests. HTML reports, richer GUI configuration and more advanced sequence diagnostics were secondary. External plugins, complete deep/multipart support and DCC integrations were explicitly removed from the MVP.

This scope protected the project from two common risks. First, migrating image libraries before defining common models could have coupled the new architecture to another backend. Second, attempting renderer-specific interpretation too early could have produced an untestable collection of naming assumptions. The core was therefore designed around data models and a small reader protocol before the production backend was selected.

3.3 Design-science research and iterative engineering method

The project is framed as Design Science Research (DSR): it addresses a relevant practice-based problem by constructing and evaluating a purposeful software artefact. Hevner et al. (2004) describe design science as the creation and evaluation of artefacts intended to solve identified organisational problems, while requiring both practical relevance and research rigour. Peffers et al. (2007) operationalise this logic as problem identification, definition of objectives, design and development, demonstration, evaluation and communication.

AOVGuard maps onto that sequence without claiming that software construction alone constitutes research. The original prototype and technical feedback identified the problem; the research questions and MoSCoW scope defined objectives; the backend-independent models, reader

boundary and rule system formed the designed artefact; real EXRs and interface workflows demonstrated it; tests, coverage, benchmarks and critical limitations evaluated it; and the repository, reports and dissertation communicate both result and method. Iteration occurred because evaluation at each stage changed later design decisions, particularly the decision to retain OpenEXR after the OpenImageIO packaging failure and to define performance at the application-reader boundary.

The implementation followed an iterative sequence:

1. record current behaviour and coverage;
2. centralise channel and brightness utilities;
3. define models and an abstract reader boundary;
4. compare candidate EXR backends through a technical spike;
5. implement structure inspection and the selected reader;
6. reorganise analysis by frame;
7. introduce rules and structured findings;
8. migrate CLI, reports and GUI to the shared backend; and
9. evaluate correctness, performance and limitations.

Each stage was supported by tests before later interfaces depended on it. Fake readers made core analysis testable without requiring an EXR decoder. Small deterministic EXR fixtures covered known channel structures, while authorised real EXRs were kept outside the repository to avoid distributing large or potentially sensitive image data. Corrupt files were created temporarily during tests rather than stored as assets.

3.4 Project origin and development evolution

AOVGuard was originally conceived around a deliberately narrow practical question: could pixel statistics help a Lighting or Pipeline TD decide quickly whether a rendered pass contained useful contribution? The project was therefore not planned initially as a complete validation framework. The initial priority was to make the core experiment visible: read an image, calculate a brightness-related metric and classify whether an AOV appeared active, nearly empty or empty. Separate simple and multilayer paths were a reasonable way to make that experiment work because OpenCV could read ordinary colour EXRs while OpenEXR exposed named channels. A CLI was added to explore automation, followed by a PySide6 interface to test whether the same idea could be accessible to artists. At that stage, however, the two interfaces were connected to analyser-specific behaviour rather than to one stable domain model.

Technical feedback changed the interpretation of the problem. Automatic EXR detection, configurable rules, selectable checks, alternative brightness weights, stronger edge-case testing and

consistent RGB handling initially appeared to be independent feature requests. When these requests were traced through the existing code, they pointed to a deeper issue: the prototype had no single representation of an inspected EXR, no shared rule result and no backend-independent analysis contract. The reported channel-order inconsistency was especially useful in this respect. Existing formulas sometimes compensated numerically for BGR and RGB, but the presence of two internal conventions made every later metric harder to reason about. Likewise, repeatedly opening frames per AOV was not only a performance concern; it revealed that the processing loop was organised around the historical implementation rather than the physical unit being delivered by a renderer.

Feature expansion was therefore paused and a behavioural baseline was established before restructuring the application. Characterisation tests recorded what the existing analysers, CLI and GUI actually did, including awkward and failing cases. This step prevented a rewrite from silently changing successful behaviour and made it possible to distinguish confirmed defects from maintainability risks. Coverage was treated as a map of unobserved behaviour rather than as the project objective. The early result of approximately 24% coverage directed attention toward the multilayer path and interfaces, but the subsequent test plan was organised around RGB/BGR reconstruction, NaN and infinity, missing channels, corrupt files, discovery ambiguity and repeated reader calls.

Table 2 summarises the principal stages through which the initial idea became the current framework. The sequence matters because each result constrained the following stage.

Table 2. Development decision trail from the initial prototype to AOVGuard 2.0.

Stage	Working question	Decision	Evidence produced
Initial pass classifier	Can pixel statistics distinguish active, nearly empty and empty AOVs?	Build separate simple and named-channel multilayer analysers with CLI and GUI access	Functional prototype, initial metrics and JSON/CSV output
Technical audit	Why are new checks difficult to add consistently?	Characterise existing behaviour before changing architecture	Code audit, baseline tests and identification of duplicated brightness calculation, RGB/BGR divergence and analyser coupling
Core design	Which concepts must be independent of the image library?	Introduce shared models, canonical RGB utilities, discovery and a small reader protocol	Fake-reader tests and backend-independent analysis cases
Reader spike	Which available backend best satisfies named-channel inspection in the target environment?	Compare OpenCV, OpenEXR and OpenImageIO; select OpenEXR pragmatically after the Windows OIIO import failure	Recorded compatibility observations and an explicit migration boundary
Processing and rules	How can all AOVs be analysed consistently without revisiting every frame per pass?	Process frame-first, update incremental accumulators and execute category-aware rule functions	Reader-call benchmark, rule-isolation tests and structured findings
Interface migration and evaluation	Can artists and pipeline scripts consume the same result without duplicating validation logic?	Move CLI, GUI, JSON, HTML and comparison workflows onto AnalysisReport	Interface-equivalence tests, real-EXR validation, contextual help and repeatability evidence

Several implementation choices emerged from this decision trail rather than from an initial fixed architecture. The `EXRReader` protocol was defined before selecting the production reader so that a dependency experiment could not dictate the core model. The rule engine began as a registry of functions instead of a plugin class hierarchy because the MVP rules were stateless and easier to test in that form. The GUI was migrated after the common API and CLI had stabilised, which kept presentation concerns from determining report structure. Deep data, complete multipart analysis and external plugins were deferred when their model and test costs became clearer. These were scope decisions, not forgotten features.

The working process also changed the meaning of the tool. The original classifier asked whether a pass looked active. AOVGuard 2.0 asks what structure was found, which interpretation was inferred, which evidence was measured, which configured rule produced a finding and what remains unknown. This shift from a label-first prototype to an evidence-first validator is the central design evolution of the project. It explains why the final contribution is not measured only by the number of added checks: the more significant result is a common vocabulary and processing path through which future checks can be added without recreating the original inconsistencies.

3.5 Reader backend spike

The reader spike compared OpenCV, OpenEXR and OpenImageIO against the MVP requirements. OpenCV successfully read simple RGB EXRs but exposed BGR arrays and did not provide named-channel inspection through `imread` (OpenCV contributors, 2025). OpenEXR 3.4.12 exposed headers, parts and named channels and integrated directly with the inspection models. OpenImageIO was technically attractive because of its broad format support, channel selection and ImageCache (OpenImageIO contributors, n.d.-a), but its Python module failed to import in the target Windows environment because of a DLL loading error. The same error remained after shortening the environment path.

OpenEXR was therefore selected as the production reader, with OpenCV retained only for legacy compatibility. This was a pragmatic decision rather than a claim that OpenEXR is universally superior. The OpenEXR Python documentation explicitly states that the high-level module prioritises simplicity and loads image data into NumPy arrays rather than providing the low-level memory control required by some large-image applications (Academy Software Foundation, n.d.-a). OpenImageIO remains a reasonable future candidate if packaging and representative performance can be demonstrated on target systems.

3.6 Evaluation design

Evaluation combines four forms of evidence. Correctness tests use known arrays, synthetic EXRs, fake readers, corrupt inputs and interface-level scenarios. Real EXR validation confirms that the tool handles an intended Maya/Arnold merged-AOV structure. A controlled benchmark compares AOV-first and frame-first organisation while preserving a checksum. Finally, maintainability and usability

are assessed through architecture, report equivalence, contextual help and a documented formative evaluation protocol.

Performance methodology follows the caution advocated by Kalibera and Jones (2013): multiple repetitions and environmental details are recorded, and timing is not generalised beyond the test conditions. Wohlin et al. (2024) similarly emphasise explicit experimental design and threats to validity. The benchmark therefore records Python, OpenEXR and NumPy versions, frame count, AOV count, resolution, repeats, reader calls, elapsed time, Python-tracked peak memory and output checksum.

The following traceability summary links each research question to its principal evidence and interpretive boundary:

RQ1 - structure-first inspection. Evidence comes from structural unit tests and two authorised 25-channel Maya/Arnold EXRs. The conclusion is limited to root RGB and named, single-part, non-deep structures represented by those tests; it is not evidence of universal EXR semantic inference.

RQ2 - shared AOV-aware rules. Evidence comes from rule isolation, preset loading, report serialization and CLI/GUI equivalence tests. The shared result is demonstrated at the software-interface level, while renderer-specific rule meaning remains outside the validated scope.

RQ3 - frame-first processing. Evidence comprises exact application-level reader-call counts, equal output checksums and a repeated timing experiment. It supports a reduction from frames multiplied by AOVs to frames at the reader abstraction, not a claim of one physical disk operation or universal speed-up.

RQ4 - sufficient engineering evidence. Evidence is triangulated across correctness tests, branch-aware coverage, invalid inputs, real files, interface equivalence and explicit threats to validity. This supports confidence in the documented prototype; it does not establish independent production certification or defect freedom.

4. System Design and Implementation

4.1 Shared architecture

The architecture centres on a common analysis pipeline. A source path is discovered once and interpreted as a single file, numbered sequence or independent comparison set. Each supported file is read through an `EXRReader` boundary, which returns a `FrameData` object containing its inspection and selected arrays. Colour AOVs are normalised to RGB and converted into metrics; technical AOVs receive per-channel diagnostics. Rules consume a shared context and return `Finding` objects. The final `AnalysisReport` is used by CLI, GUI, JSON, HTML and baseline comparison. Figure 2 illustrates this separation and the common report model consumed by every interface.

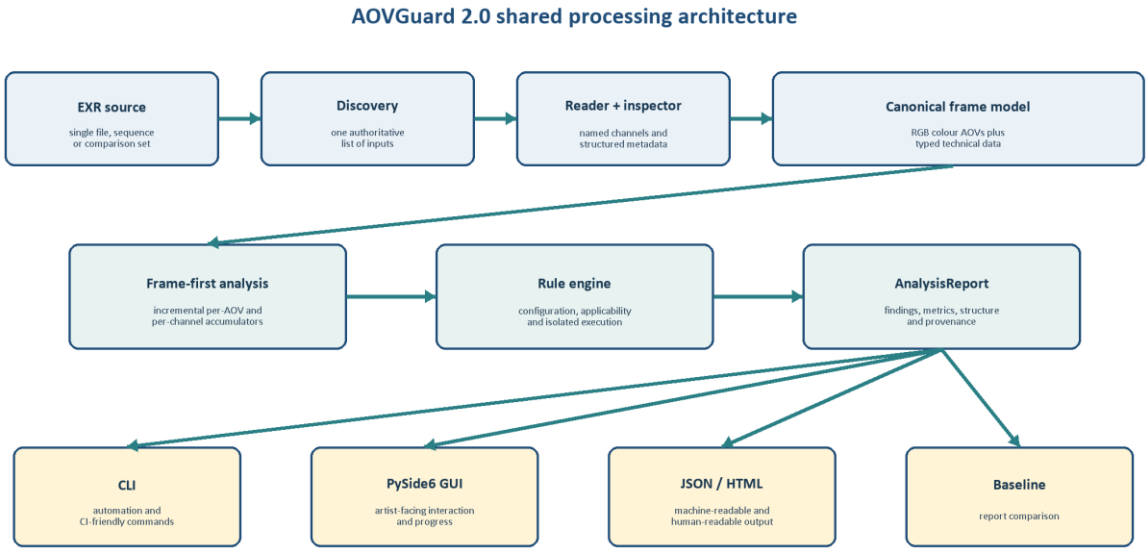


Figure 2. AOVGuard 2.0 shared architecture. CLI, GUI and report formats consume the same analysis model.

The separation addresses several risks in the original design. Image-library behaviour is contained at the IO boundary. Discovery cannot disagree with progress reporting because the same immutable tuple of paths is used throughout. Rules do not write directly to the GUI. JSON and HTML do not re-run analysis. This reduces duplication and makes equivalence testable.

4.2 Models and reader boundary

The minimum shared models include `AnalysisOptions`, `FileInspection`, `AOVDescriptor`, `FrameData`, `MetricSet`, `Finding`, `SequenceInfo` and `AnalysisReport`. `FileInspection` records physical structure; `AOVDescriptor` records a logical interpretation; `Finding` records validation evidence; and `AnalysisReport` records what was discovered, processed and rejected.

The reader boundary is defined as a Python `Protocol` with `inspect(path)` and `read_frame(path, requested_aovs)` operations. Structural typing was selected instead of an abstract base class because fake test readers can satisfy the interface without inheritance. The interface remains

deliberately small: scanline, tile, cache and lifecycle details stay inside the backend until a demonstrated requirement justifies exposing them.

The production `OpenEXRReader` uses `OpenEXR.File`. Header-only inspection is available for the explicit inspection command. During analysis, however, the `FrameData` object carries its own inspection so that a frame does not require a second application-level header read. The first part's channels are grouped and converted to float32 NumPy arrays. Unsupported deep or multipart structures produce an explicit reason rather than entering pixel analysis.

4.3 Structure inspection and AOV inference

The inspector groups root `R/G/B/A` channels into a `beauty` AOV and dotted channels by their prefix. Channel suffixes are interpreted case-insensitively for common roles such as R, G, B, A, X, Y and Z. Broad categories are inferred from both structure and conventional names. Three colour roles produce a colour descriptor, one channel can become scalar or depth, and recognised triplets can become vectors.

This approach supports the project's real Maya/Arnold examples, including `albedo.R/G/B`, `diffuse_direct.R/G/B`, `specular_direct.R/G/B`, `emission.R/G/B`, `N.R/G/B`, `P.R/G/B` and `Z.R`. It is not a universal semantic parser. A custom `motion` pass, for example, may contain two vector components, while cryptomatte or light-group naming can be more complex. Ambiguous structures are therefore assigned `unknown`, and the design leaves room for preset overrides and renderer-specific adapters.

Table 3 summarises the current analysis policy.

Table 3. AOV category inference and current analysis policy.

Category	Typical examples	Current safe analysis	Current exclusions
Colour	beauty, albedo, diffuse, specular, emission	RGB brightness, non-black ratio, NaN/Inf, empty/near-empty	No automatic colour-space interpretation
Vector	N, P	per-channel min/mean/max, NaN/Inf, negative counts	No normal-length or coordinate-space semantics
Depth	Z	per-channel min/mean/max, NaN/Inf	No camera-range or focus-plane semantics
Mask/scalar	alpha, IDs, custom masks	per-channel diagnostics, configurable constant checks	Constant or empty may be valid by intent
Unknown	unconventional channel structures	structure reported for review	Not forced through colour rules

4.4 Canonical RGB and centralised brightness

The reader orders colour by channel suffix, not array position inherited from a library. RGB data are stacked in the order R, G, B at the IO boundary. Alpha is structural information and is not included in

the brightness formula. Monochrome, depth and vector data are not expanded into false RGB images.

`core/luminance.py` defines the Rec.709 and Rec.601 presets and validates custom weights. Weights must contain three finite values and have a positive sum; the configured policy ensures deterministic behaviour. Centralisation removes the duplicated formula from the original simple and multilayer paths and makes unit tests independent of EXR files. Tests use synthetic pure red, green and blue pixels to demonstrate both channel order and expected weighted results.

The implementation uses finite masks before aggregation. NaN and infinity counts are preserved as evidence, while valid-pixel metrics are calculated without silently classifying non-finite arrays as active. Signed and absolute statistics are separated where needed so that negative values do not cancel positive contributions unnoticed. Brightness-related rules are restricted to colour descriptors.

4.5 Discovery, source interpretation and sequences

Frame discovery is implemented once in `discovery/frame_discovery.py`. Options include a filename pattern, recursive search and bounded depth. The resulting tuple is used for processing, progress, sequence checks and reporting. This prevents the earlier situation in which one search counted frames while another selected the files.

Discovery and interpretation are separate decisions. A folder may represent a numbered sequence such as `shot.1001.exr`, or it may contain two independent still renders intended for comparison. Auto mode infers a single file, sequence or comparison set. Users can select Sequence or Comparison explicitly when filenames are ambiguous. Multiple numbered sequences are rejected unless the user deliberately permits them. The report records the resolved source kind so that sequence gaps are not incorrectly reported for independent look comparisons.

The sequence checker parses the final numerical token, groups consistent name patterns and detects missing frames, duplicate numbers and mixed padding. This is useful without decoding pixels, but it remains a convention rather than a show-aware parser. Filenames with several meaningful numeric tokens or custom version syntax may require an explicit pattern or future token template.

4.6 Frame-first incremental processing

The original multilayer organisation could process one AOV across every frame before moving to the next AOV. For N frames and A AOVs, this leads to approximately N multiplied by A application-level reader calls. The new loop discovers frames once, reads a frame, extracts all supported AOVs, updates accumulators and releases the frame arrays before continuing. Figure 3 contrasts the two traversal strategies.

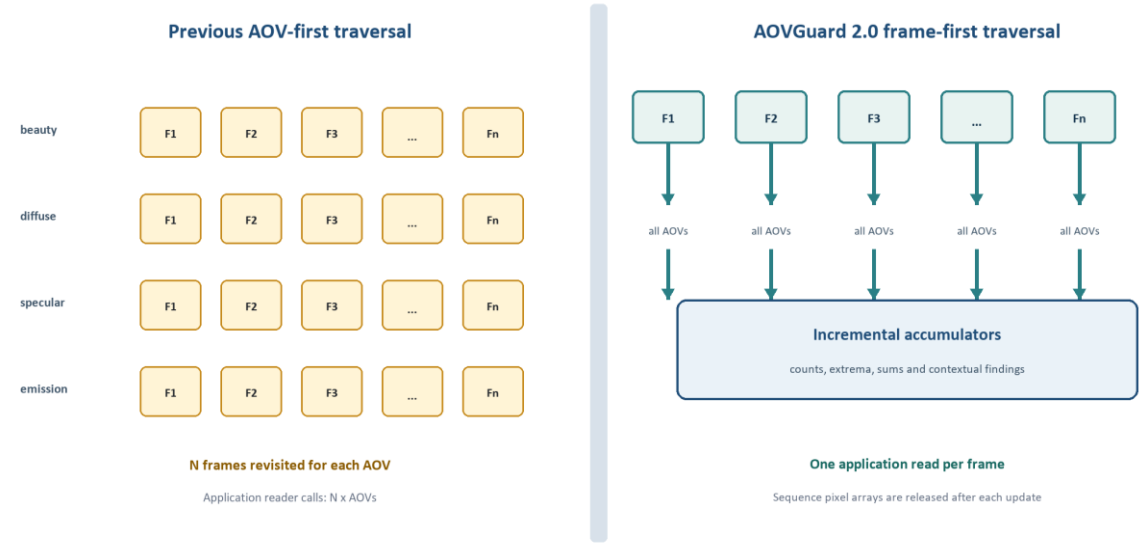


Figure 3. Processing reorganisation. AOV-first revisits frames for each pass, whereas frame-first updates all pass accumulators from one application-level frame read.

Frame-first processing pseudocode

```
discover frames -> for each frame: read once -> inspect AOVs -> calculate metrics -> update accumulators -> apply rules -> finalise sequence metrics -> build AnalysisReport
```

Application-level reader-call model

```
C_AOV-first ~= N x A
```

```
C_frame-first = N
```

where `N` is the number of frames and `A` is the number of AOVs.

Accumulators retain counts, sums, extrema, finite-value information and per-frame metric summaries rather than a full sequence of image arrays. This bounds Python-visible retained image state primarily to the current frame. Cross-frame analysis calculates median, median absolute deviation, consecutive change and outlier counts from per-frame metrics. AOVs that appear or disappear and resolution changes generate consistency findings associated with the affected frame.

"One read per frame" is deliberately defined at the application interface. The high-level OpenEXR Python object may perform multiple internal operations and may decode more data than the requested AOV set. The benchmark instruments AOVGuard reader calls, not operating-system opens. This distinction prevents the architectural claim from being confused with low-level IO behaviour that the current API does not expose.

4.7 Configurable rule system

The MVP rule system uses data definitions and a function registry rather than a hierarchy of rule classes. A `RuleDefinition` contains an identifier, enabled flag, severity, parameter mapping and supported AOV categories. The registry maps an identifier to a validation function. Each function receives a shared context and returns a list of `Finding` objects.

This design keeps stateless checks concise and easy to test. Class-based rules would become useful if future checks required persistent state, expensive initialisation, external resources or third-party plugin lifecycles. Those needs are not present in the MVP, so an inheritance framework would add complexity without current benefit.

Rules can be loaded from TOML or JSON. TOML is convenient for versioned, human-edited presets within a Python project; JSON remains useful for generated configurations and pipeline interchange. CLI and GUI both translate user selections into the same rule definitions. The engine catches exceptions around each rule and records a `rule_error` finding, allowing other checks to continue while ensuring failures are visible.

The built-in catalogue is shown in Table 4.

Table 4. Built-in configurable validation rule catalogue.

Rule	Purpose	Default interpretation
nan_inf	Count NaN, positive Inf and negative Inf	Error because non-finite values can propagate
empty_aov	Detect no visible colour contribution	Warning because an intentionally unused pass may be empty
near_empty_aov	Detect contribution below a configured ratio	Warning requiring contextual review
missing_aov	Compare required AOV names with discovered structure	Configurable error/warning by delivery preset
missing_channels	Check required channel roles	Error when a declared structure is incomplete
negative_values	Report negative finite values	Contextual warning or information, category-aware
constant_channel	Identify zero dynamic range	Information or warning; may be valid for masks
resolution_consistency	Compare frame dimensions	Error for a single intended sequence
aov_consistency	Compare AOV/channel structure across frames	Error or warning according to delivery policy

The project intentionally avoids a universal highlight-clipping error. Scene-linear HDR values above one are valid, so any threshold must be show-configurable and interpreted as a metric or warning. The same reasoning applies to hot pixels and shadow thresholds, which remain possible extensions after representative data and user requirements are available.

4.8 Reports and source comparison

`AnalysisReport` is the canonical result. It records schema version, source, source kind, options, discovered frames, processed frames, failed frames, inspections, aggregate and per-frame metrics, technical channel metrics, sequence data, findings and status. JSON serialization uses

`allow_nan=False`; non-finite metric fields become `null`, while counts remain explicit. This conforms to the interoperability constraints in RFC 8259 (Bray, 2017).

The HTML report is generated from the same model and escapes untrusted text. It is a human-readable secondary representation rather than a second source of truth. Baseline comparison reads two canonical JSON reports and identifies added, removed, changed and unchanged AOV metrics together with new and resolved findings. This supports regression-style render review without embedding source images in the repository.

4.9 CLI and GUI

The current CLI provides `inspect-structure`, `analyze`, `check-sequence` and `compare-reports`, with legacy commands retained temporarily for backward comparison. The main `analyze` command supports preset files, source mode, discovery options, JSON/HTML output and strict exit behaviour. PASS and WARNING return success by default, while `--fail-on-warning` allows publishing or CI policies to treat warnings as failures.

The PySide6 GUI exposes the same backend. It provides file/folder selection, inspection, source settings, rule checkboxes, brightness-weight selection, progress, cooperative cancellation, findings, metrics, per-sample and technical tabs, sequence discovery, comparison, contextual tooltips and JSON/HTML export. Analysis runs in a worker associated with `QThread`, which keeps the GUI event loop responsive; Qt recommends moving long-running or blocking work away from the GUI thread (Qt Project, n.d.). Cancellation is cooperative between frames rather than forcibly terminating a decoder mid-read.

The interface deliberately presents definitions on hover for controls, AOV categories and metrics. It also separates aggregate findings from per-frame evidence and opens the most relevant result tab automatically. These changes improve accessibility for artists without moving validation logic into presentation code. Figure 4 shows the resulting report-oriented interface with a contextual warning selected.

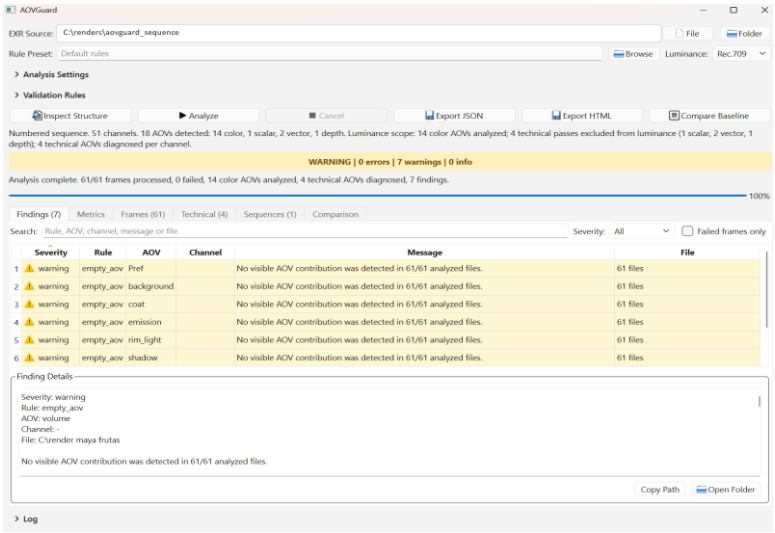


Figure 4. Current AOVGuard 2.0 GUI showing automatic structure context, sequence findings and shared report-oriented results. The displayed source path has been sanitised; EXR image data remain external to the repository.

5. Evaluation

5.1 Automated correctness and coverage

The test suite covers core models, channel handling, luminance presets, custom-weight validation, reader behaviour, inspection heuristics, discovery, sequence grouping, rule execution, rule failure isolation, strict JSON, HTML escaping, report comparison, CLI exit behaviour, GUI state and error handling. Synthetic arrays establish exact expected values. Small generated EXRs exercise RGB, RGBA, named multilayer channels, missing channels and corrupt input. Fake readers count calls and inject later-frame failures without depending on filesystem behaviour.

The final verification was executed on 14 August 2026 using Python 3.12.12 on Windows. All 229 tests passed in 4.75 seconds. Coverage measured 3,089 statements and 878 branches across the `aovguard` package, with 95.24% combined coverage and a 90% enforced threshold. Critical modules for channel utilities, luminance, models, status, the OpenEXR reader, rule engine, loader and sequence checker reached 100% in that run. The broader Rules group shown in Figure 5 reached 99%, because that grouped result also includes surrounding rule-package code beyond the individual 100% modules. Core analysis reached 91%, the CLI 95%, and the GUI module 91%. Figure 5 summarises the verified reach of selected module groups without treating coverage as proof of correctness.

These results answer part of RQ4: the suite exercises a broad proportion of the implementation and includes risk-focused edge cases. They do not prove absence of defects. Inozemtseva and Holmes (2014) warn against treating coverage as a proxy for fault-detection effectiveness. The stronger evidence comes from combining coverage with known-value assertions, branch cases, invalid inputs, interface equivalence and real-file evaluation.

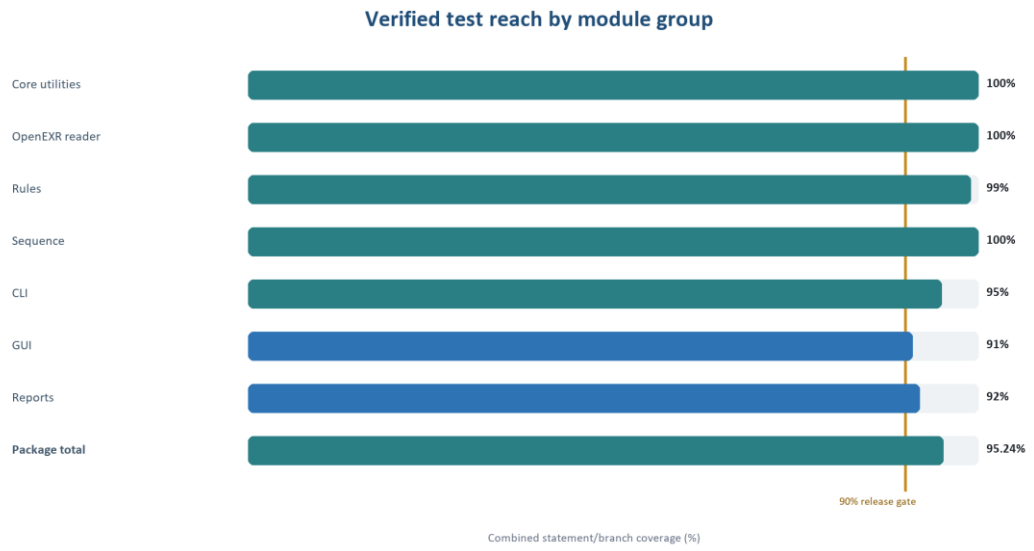


Figure 5. Verified coverage for selected AOVGuard module groups on Python 3.12. Individual rule-engine and loader modules reached 100%, while the broader Rules group shown here reached 99%. The total combined statement/branch result was 95.24%; coverage is presented as test reach, not defect-free proof.

5.2 Validation with authorised real EXRs

Real workflow validation used two external multilayer EXR files exported from Maya/Arnold. They were not committed to the repository. One inspected file was 1920 by 1080 pixels, single-part and non-deep, with 25 channels grouped into eight AOV descriptors: beauty, N, P, Z, albedo, diffuse_direct, emission and specular_direct. Five were classified as colour, N and P as vectors, and Z as depth.

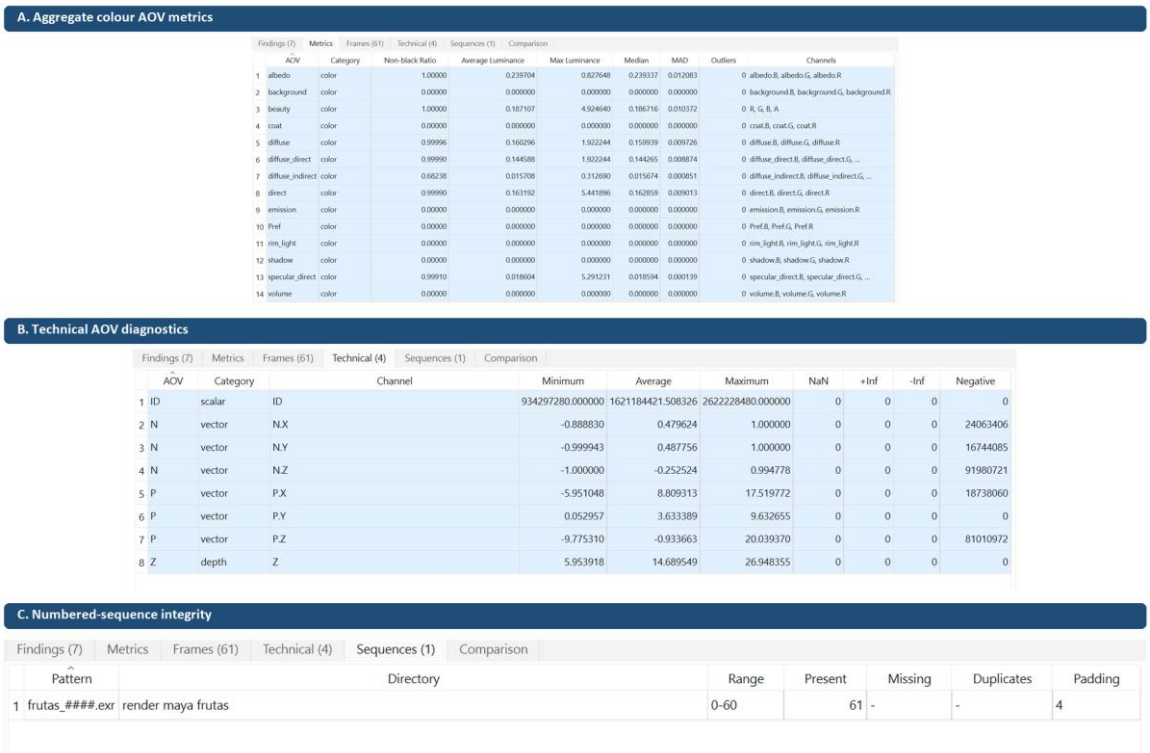
The automatic workflow analysed both files as independent comparison samples. It processed two of two files, reported zero read failures, analysed five colour AOVs and diagnosed three technical AOVs per channel. The `empty_aov` rule produced one warning because emission had no visible contribution in both files. N and P retained expected negative component values as technical metrics rather than being incorrectly classified by colour brightness rules. This result supports RQ1 for the tested single-part named-channel structure.

The evidence is intentionally bounded. Two files from one renderer and workflow do not establish compatibility with all Arnold versions, other renderers, multipart layouts, cryptomatte naming or production colour configurations. The result demonstrates a representative use case, not universal EXR interpretation.

5.3 Additional sequence workflow validation

An additional authorised Maya/Arnold workflow test used the numbered sequence `frutas_0000.exr` to `frutas_0060.exr`. AOVGuard discovered a continuous 61-frame range with four-digit padding, no missing frame numbers and no duplicates. Inspection identified 51 named channels grouped into 18 AOV descriptors: 14 colour, one scalar, two vector and one depth descriptor. All 61 files were processed and no frame read failed. Seven warnings identified colour AOVs with no visible contribution across the sequence; these are contextual validation results rather than decoder failures.

Figure 6 records the aggregate colour metrics, per-channel technical diagnostics and sequence-integrity evidence from this run. This experiment provides additional workflow evidence for a real numbered render sequence. It is reported separately from the controlled 12-frame, five-AOV benchmark because the two evaluations answer different questions and use different datasets.



A. Aggregate colour AOV metrics

Findings (7) | Metrics | Frames (61) | Technical (4) | Sequences (1) | Comparison

	AOV	Category	Non-black Ratio	Average Luminance	Max Luminance	Median	MAD	Outliers	Channels
1	albedo	color	1.00000	0.239704	0.827648	0.239337	0.012083	0	albedo.B, albedo.G, albedo.R
2	background	color	0.00000	0.000000	0.000000	0.000000	0.000000	0	background.B, background.G, background.R
3	beauty	color	1.00000	0.187107	4.924640	0.186716	0.010372	0	R, G, B, A
4	coat	color	0.00000	0.000000	0.000000	0.000000	0.000000	0	coat.B, coat.G, coat.R
5	diffuse	color	0.99996	0.160296	1.922244	0.159939	0.009726	0	diffuse.B, diffuse.G, diffuse.R
6	diffuse_direct	color	0.99990	0.144588	1.922244	0.144265	0.008874	0	diffuse_direct.B, diffuse_direct.G, ...
7	diffuse_indirect	color	0.68238	0.015708	0.312690	0.015674	0.000851	0	diffuse_indirect.B, diffuse_indirect.G, ...
8	direct	color	0.99990	0.163192	5.441896	0.162859	0.009013	0	direct.B, direct.G, direct.R
9	emission	color	0.00000	0.000000	0.000000	0.000000	0.000000	0	emission.B, emission.G, emission.R
10	Pref	color	0.00000	0.000000	0.000000	0.000000	0.000000	0	Pref.B, Pref.G, Pref.R
11	rim_light	color	0.00000	0.000000	0.000000	0.000000	0.000000	0	rim_light.B, rim_light.G, rim_light.R
12	shadow	color	0.00000	0.000000	0.000000	0.000000	0.000000	0	shadow.B, shadow.G, shadow.R
13	specular_direct	color	0.99910	0.018604	5.291231	0.018594	0.000139	0	specular_direct.B, specular_direct.G, ...
14	volume	color	0.00000	0.000000	0.000000	0.000000	0.000000	0	volume.B, volume.G, volume.R

B. Technical AOV diagnostics

Findings (7) | Metrics | Frames (61) | Technical (4) | Sequences (1) | Comparison

	AOV	Category	Channel	Minimum	Average	Maximum	NaN	+Inf	-Inf	Negative
1	ID	scalar	ID	934297280.000000	1621184421.508326	2622228480.000000	0	0	0	0
2	N	vector	N.X	-0.888830	0.479624	1.000000	0	0	0	24063406
3	N	vector	N.Y	-0.999943	0.487756	1.000000	0	0	0	16744085
4	N	vector	N.Z	-1.000000	-0.252524	0.994778	0	0	0	91980721
5	P	vector	P.X	-5.951048	8.809313	17.519772	0	0	0	18738060
6	P	vector	P.Y	0.052957	3.633389	9.632655	0	0	0	0
7	P	vector	P.Z	-9.775310	-0.933663	20.039370	0	0	0	81010972
8	Z	depth	Z	5.953918	14.689549	26.948355	0	0	0	0

C. Numbered-sequence integrity

Findings (7) | Metrics | Frames (61) | Technical (4) | Sequences (1) | Comparison

	Pattern	Directory	Range	Present	Missing	Duplicates	Padding
1	frutas_####.exr	render maya frutas	0-60	61	-	-	4

Figure 6. Additional Maya/Arnold sequence validation: aggregate colour metrics, technical AOV diagnostics and sequence integrity for 61 EXRs. The sequence contained 51 channels and 18 inferred AOV descriptors; 61 of 61 frames were processed with no missing frames, duplicates or read failures.

5.4 Frame-first benchmark

The controlled benchmark used 12 generated frames, five AOVs, 320 by 180 resolution and 30 recorded repetitions per strategy on Windows 11 build 26200, using a 13th Gen Intel Core i9-13900HX with 32 logical processors, Python 3.12.12, NumPy 2.4.6 and OpenEXR 3.4.12. Each strategy received one unrecorded warm-up, and execution order alternated on every repetition to reduce systematic first-run and cache-order bias. The legacy-style AOV-first loop made 60 reader calls; frame-first made 12, a fivefold reduction corresponding to the number of AOVs. The pixel checksum was identical at 5,365,412.494333796.

Median elapsed time changed from 0.4387 seconds to 0.1181 seconds, a 3.72 times speed-up in that environment. The interquartile ranges were 0.0730 and 0.0369 seconds respectively, and the descriptive minima were 0.2571 and 0.0617 seconds. Python-tracked peak memory increased from approximately 5.55 MB to 11.08 MB in the frame-first case. This trade-off is plausible because all five AOV arrays for the current frame are available together instead of one AOV at a time. Figure 7 presents the call-count, timing and Python-tracked memory comparison.

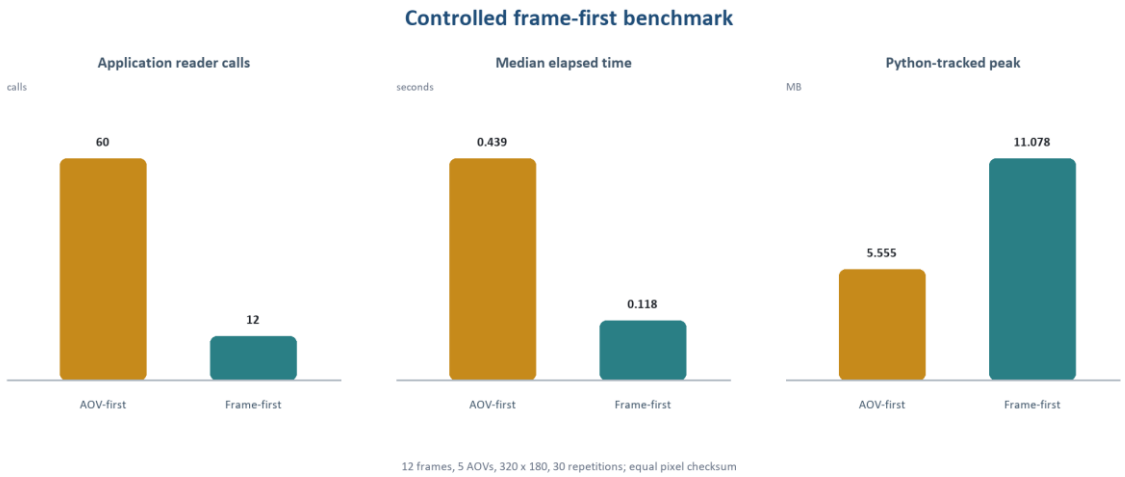


Figure 7. Controlled benchmark results. Frame-first reduced application-level reader calls and median time, while Python-tracked peak memory increased.

The benchmark supports RQ3 only under its documented conditions. `tracemalloc` does not capture native allocations made by OpenEXR, operating-system caching was not fully isolated, the resolution is small, and one workstation cannot establish a general performance model. The OpenEXR high-level Python API may decode whole-file data at construction. The strongest result is therefore architectural and exact: reader calls decreased from N multiplied by A to N and output checksums matched. The timing distribution is encouraging but requires larger, controlled real-EXR experiments on additional hardware before a production claim can be made.

5.5 CLI/GUI equivalence and usability protocol

Integration tests construct the same options through CLI and GUI adapters and compare canonical payloads. This demonstrates that interfaces share analysis rather than reproducing rule logic. GUI tests cover essential state changes, worker completion, errors, cancellation boundaries, rule selection, output population and tooltips. They deliberately avoid pursuing every paint or platform event for coverage alone.

A small formative usability protocol has been prepared but no participant results are claimed in the current dissertation. Proposed participants are three to five people familiar with lighting, compositing, rendering or pipeline work. Tasks include selecting a real EXR, interpreting detected AOV categories, explaining the emission warning, changing the brightness-weight configuration, locating a missing frame and exporting JSON. Measures include completion without assistance, time, interpretation errors and confidence. University ethics requirements must be confirmed before recruitment or recording. If approval or time is insufficient, expert review or clearly labelled informal feedback should replace a formal user study.

5.6 Maintainability, repeatability and reproducibility support

Maintainability evidence includes removal of duplicated brightness logic, one reader boundary, shared CLI/GUI models, a function registry for rules, versioned report output and a documented source-discovery policy. A new preset can change severities and parameters without changing the

engine; a new stateless rule can be registered without modifying interface code. This is consistent with the project's objective of reducing coupling, although `ui.py` remains a large module and is a candidate for further decomposition.

The terminology follows the ACM distinction between repeatability in the same team and experimental setup, replicability by a different team using the same artefacts, and reproducibility through an independently developed measurement (Association for Computing Machinery, n.d.). The recorded commands, locked dependencies, tests and benchmark demonstrate repeatability in the documented development environment and provide support for future independent replication. They do not prove external reproducibility because no independent team repeated the study and the authorised real EXRs cannot be distributed.

This support is provided by `pyproject.toml`, `uv.lock`, a Windows/Ubuntu GitHub Actions matrix for Python 3.11 and 3.12, a 90% coverage gate, benchmark scripts and raw JSON results. The clean release process excludes virtual environments, caches and every EXR file. Sample reports are anonymised. The VFX Reference Platform provides useful industry context for dependency coordination, although AOVGuard is not yet packaged for embedding in DCC applications (Visual Effects Society Technology Committee, 2026).

Table 5 summarises the evaluated outcomes.

Table 5. Summary of evaluation evidence, results and remaining qualifications.

Objective	Evidence	Result	Remaining qualification
Automatic inspection	Real 25-channel Arnold EXR plus structural tests	Eight AOVs categorised without manual mode	Naming remains heuristic; deep/multipart unsupported
RGB consistency	Pure-channel tests and named-channel reconstruction	One canonical RGB convention	Input colour space is not inferred
Rule consistency	Shared context, CLI/GUI equivalence tests	Same findings model across interfaces	Renderer semantics remain limited
Frame-first processing	Instrumented 12-frame benchmark	60 to 12 reader calls; checksum match	Native IO and larger workloads need measurement
Reliability	229 tests and invalid-input cases	All tests passed in final run	Coverage is not proof of no defects
Reporting	Strict JSON and escaped HTML tests	Canonical schema version 1.0	Formal schema publication/version migration is future work

6. Critical Discussion

6.1 Interpretation of the contribution

The evidence supports the claim that AOVGuard 2.0 is a substantially more coherent validation prototype than its first version. Manual mode selection has been removed from the main workflow, RGB handling and brightness are centralised, discovery is consistent, rules are configurable, file reads are organised by frame, and both interfaces consume one backend. These improvements directly address the original technical feedback.

The project should not be described as a replacement for OpenImageIO, OpenEXR tools, a compositor, or a studio QC platform. Its value lies in integration and presentation. It translates named channels and image statistics into AOV-aware evidence, offers a compact reference implementation, and exposes decisions that could otherwise remain hidden in ad hoc scripts. This is appropriate for an MSc contribution because it combines domain knowledge, architecture, implementation and evaluation rather than claiming a new image format or statistical method.

6.2 Correctness and false-positive risk

Some rules are objectively defensible. A missing required channel, unreadable file, dimension change or non-finite value can be reported with clear evidence. Other conditions are contextual. An empty emission AOV may be expected in a scene without emissive surfaces. A constant mask may be valid. Negative values can be valid and expected in position and normal components, and may also occur in colour-processing pipelines. Very high scene-linear values are not automatically clipping.

For this reason, the report distinguishes PASS, WARNING and FAIL, permits severity configuration and records metrics rather than only labels. Presets should encode delivery policy, not artistic truth. Future work should add an explicit provenance field to AOV categories and allow show configurations to override both inferred type and rule applicability. A renderer-aware vocabulary could recognise aliases without making the core dependent on one renderer.

6.3 Colour and brightness limitations

Canonical RGB solves channel-order inconsistency, but not colour interpretation. AOVGuard currently assumes that a selected coefficient set is meaningful for the incoming values. OpenEXR defines a standard `chromaticities` attribute, yet files may omit it, and AOVs may use a show working space such as ACEScg (Academy Software Foundation, n.d.-d). Rec.709 and Rec.601 weights should therefore be described as configurable brightness metrics. A future colour-aware implementation should inspect declared chromaticities or accept an explicit working-space preset, derive appropriate linear luminance coefficients, and distinguish physical luminance from non-linear luma and the current brightness proxy. This is more technically defensible than adding more fixed formulas without input-colour context.

6.4 IO and performance limitations

Frame-first processing reduces repeated calls made by AOVGuard and avoids retaining a full sequence, but it is not lazy channel IO in the strongest sense. The high-level OpenEXR Python API prioritises simple full-array access. Requested AOV filtering controls the arrays returned by the reader abstraction, but may not prevent the native library from loading all channel data when `OpenEXR.File` is constructed. Large 4K or 8K multilayer frames may therefore produce significant native memory use.

OpenImageIO remains relevant because its ImageCache supports demand-driven access to large image collections (OpenImageIO contributors, n.d.-a). The Windows import failure prevented fair production adoption during this project. Future evaluation should repeat the spike on Linux and a clean Windows packaging environment, then compare named-channel reads, partial access, native peak memory and deployment complexity. Maintaining two production readers is justified only if measured compatibility or performance benefits outweigh the testing burden.

6.5 Threats to validity

Construct validity: Non-black ratio and weighted brightness are proxies for activity, not direct measures of usefulness. Rule thresholds may not represent artistic intent.

Internal validity: Filesystem cache, background processes and Python/native allocation boundaries affect the benchmark. The checksum confirms output equivalence for the fixture but not every path.

External validity: Real-file validation used two Maya/Arnold comparison samples and one 61-frame Maya/Arnold sequence from a limited set of scenes and channel conventions. Results cannot be generalised to all renderers, studios, resolutions or EXR features.

Conclusion validity: Thirty repetitions provide a more stable descriptive distribution than the earlier three-run baseline, but the small generated fixture and single workstation still make the 3.72 times timing ratio contextual rather than universal. Larger real workloads, additional systems and inferential analysis would strengthen the result.

Evaluation bias: Most development tests were written alongside the implementation. Independent user or expert evaluation has not yet been completed. The thesis must keep planned usability evaluation separate from observed results.

Data governance: Real EXRs may contain confidential imagery or metadata. The project therefore excludes EXR files from the repository and retains only anonymised reports and authorised screenshots. This protects data but prevents exact third-party replication of the real-file result.

6.6 Research-question synthesis and lessons learned

RQ1 is supported within a defined structural boundary. Structure-first inspection removed the manual simple/multilayer choice and correctly represented the tested root RGB and named-channel EXRs. It did not solve universal semantic inference: unknown naming, deep samples and complete

multipart processing remain explicit unsupported or ambiguous cases. The useful result is therefore a richer description that guides analysis and fails clearly, not a perfect classifier.

RQ2 is strongly supported at the application boundary. CLI, GUI and report exporters consume the same models and rule outputs, and integration tests compare their canonical payloads. This removes a confirmed source of divergence in the first version. The remaining weakness is policy depth: shared execution does not make a generic threshold appropriate for every renderer, show or AOV category.

RQ3 is supported architecturally and conditionally in timing. The reader-call reduction and equal checksum are deterministic for the benchmark design. The repeated timing distribution also favours frame-first processing on the recorded workstation, but the higher Python-tracked peak memory and unmeasured native allocation show that the improvement is a trade-off rather than a free optimisation. This distinction is more informative than reporting speed-up alone.

RQ4 is answered through triangulation. No individual measure is sufficient: coverage indicates exercised code, known-value tests establish selected numerical behaviour, corrupt inputs exercise failure paths, real EXRs demonstrate selected intended workflows, equivalence tests support interface consistency, and the benchmark evaluates one architectural claim. Together they justify confidence in a focused prototype while leaving production readiness, external replication and broad renderer compatibility unproven.

Several decisions worked particularly well. Pausing feature expansion to write characterisation tests reduced regression risk. Defining models and a small reader protocol before choosing a backend kept the core testable. A function registry was adequate for stateless MVP rules and avoided an unnecessary plugin hierarchy. Moving the GUI after the shared API stabilised preserved one result model across interfaces.

Other decisions were unsuccessful or qualified. OpenImageIO was technically promising but could not be imported reliably in the target Windows environment, so adopting it would have weakened delivery confidence. Frame-first processing increased Python-tracked peak memory. Heuristic AOV typing remains intentionally uncertain, and the large `ui.py` module shows that backend decoupling did not complete every internal refactor. A formal participant usability study was planned but not conducted, so no user-performance claim is made.

The most transferable lesson is that evidence boundaries should shape architecture. Named channels are safer than positional assumptions; contextual findings are safer than artistic verdicts; a small protocol is safer than committing the core to an experimental dependency; and precise claims about application-level reads are safer than implying physical disk behaviour. These lessons explain both the achieved scope and the work deliberately left outside it.

6.7 Future technical direction

The next development priority should be stronger semantic and performance evidence rather than further GUI expansion. Recommended work includes colour-space-aware brightness, explicit category overrides, renderer-specific naming adapters, technical rules for normal-vector length and

depth validity, native-memory profiling, larger real sequences, and a repeatable cross-platform reader comparison. A packaged executable would improve accessibility only after dependency and licence behaviour are verified.

Deep and multipart support should remain separate milestones. Detection is already useful because it prevents silent misuse. Full support would require part-aware report models, per-part channel namespaces, deep sample semantics and substantially larger tests. External plugins should likewise be deferred until built-in rule APIs and report versioning have stabilised.

7. Conclusion

This dissertation has presented the design, implementation and evaluation of AOVGuard 2.0, a modular EXR and AOV validation prototype for VFX pipelines. The project began with a functional but coupled application and reorganised it around automatic structure inspection, canonical RGB data, centralised configurable brightness, one discovery result, frame-first processing, AOV-aware rules and a shared report model.

RQ1 is supported for the tested MVP structures: root RGB and named single-part multilayer EXRs can be inspected and categorised without manual simple/multilayer selection. RQ2 is supported by the shared rule context, structured findings and CLI/GUI equivalence tests. RQ3 is supported at the application boundary: the controlled benchmark reduced reader calls from 60 to 12 and preserved the checksum, with a measured median speed-up under the recorded conditions. RQ4 is addressed through layered evidence rather than a single metric: 229 tests, 95.24% branch-aware coverage, invalid-input cases, real EXR validation, a repeatable benchmark and an explicit limitations analysis.

The most important technical achievement is consistency. Colour data have one internal order; brightness-related calculations have one implementation; discovered files have one source of truth; interfaces have one backend; and reports have one canonical model. The most important limitation is semantic context. The application can measure and report, but it cannot infer artistic intent, universal AOV meaning or physically correct colour-space luminance from channel names alone.

AOVGuard 2.0 is therefore suitable as a focused MSc artefact. It demonstrates domain-informed engineering, measurable architectural improvement and critical awareness of what the evidence does not establish. A realistic production path would build on this foundation with representative show data, colour and renderer presets, native-resource profiling and carefully scoped integration, while preserving the principle that automated validation assists rather than replaces human judgement.

References

Academy of Motion Picture Arts and Sciences. (2025a). *ACES documentation: Overview of ACES encodings*. https://docs.acescentral.com/encodings/overview/

Academy of Motion Picture Arts and Sciences. (2025b). *ACEScg specification*. https://docs.acescentral.com/encodings/acescg/

Academy Software Foundation. (n.d.-a). *The OpenEXR Python module*. https://openexr.com/en/latest/python.html

Academy Software Foundation. (n.d.-b). *OpenEXR tools*. https://openexr.com/en/latest/tools.html

Academy Software Foundation. (n.d.-c). *Technical introduction to OpenEXR*. https://openexr.com/en/latest/TechnicalIntroduction.html

Academy Software Foundation. (n.d.-d). *Standard attributes*. https://openexr.com/en/latest/StandardAttributes.html

Alway, C., Nagle, P., & Keech, G. (2018). Just get on with it: A managed approach to AOV manipulation. In *Proceedings of the ACM SIGGRAPH Digital Production Symposium*. Association for Computing Machinery. https://doi.org/10.1145/3233085.3233091

Ammann, P., & Offutt, J. (2017). *Introduction to software testing* (2nd ed.). Cambridge University Press. https://doi.org/10.1017/9781316771273

Association for Computing Machinery. (n.d.). *Artifact review and badging*. https://www.acm.org/publications/policies/artifact-review-and-badging-current

Autodesk. (2024a). *Arnold for Maya user guide: AOVs*. https://help.autodesk.com/cloudhelp/2024/ENU/AR-Maya/files/am-Arnold_for_Maya_User_Guide/render-settings/arnold_for_maya_render_settings_aovs_html.html

Autodesk. (2024b). *Arnold user guide: EXR driver*. https://help.autodesk.com/cloudhelp/ENU/AR-Core/files/ac-output-aovs/arnold_user_guide_ac_output_aovs_ac_exr_html.html

Bray, T. (2017). *The JavaScript Object Notation (JSON) data interchange format* (RFC 8259). Internet Engineering Task Force. https://doi.org/10.17487/RFC8259

Brinkmann, R. (2008). *The art and science of digital compositing* (2nd ed.). Morgan Kaufmann.

Cohen Bengio, J., Ogino, K., & Robson, B. (2018). A holistic approach to asset quality and efficiency. In *ACM SIGGRAPH 2018 Talks* (pp. 1-2). Association for Computing Machinery. https://doi.org/10.1145/3214745.3214791

Debevec, P. E., & Malik, J. (1997). Recovering high dynamic range radiance maps from photographs. In *Proceedings of SIGGRAPH 97* (pp. 369-378). Association for Computing Machinery. https://doi.org/10.1145/258734.258884

Haddon, J., Kaufman, A., Minor, D., Dresser, D., Imanishi, I., & Nogueira, P. (2016). Gaffer: An open-source application framework for VFX. In *Proceedings of the ACM SIGGRAPH Digital Production Symposium* (pp. 31-42). Association for Computing Machinery. https://doi.org/10.1145/2947688.2947696

Hevner, A. R., March, S. T., Park, J., & Ram, S. (2004). Design science in information systems research. *MIS Quarterly, 28*(1), 75-105. https://aisel.aisnet.org/misq/vol28/iss1/6/

IEEE. (2019). *IEEE standard for floating-point arithmetic* (IEEE Std 754-2019). https://doi.org/10.1109/IEEESTD.2019.8766229

Inozemtseva, L., & Holmes, R. (2014). Coverage is not strongly correlated with test suite effectiveness. In *Proceedings of the 36th International Conference on Software Engineering* (pp. 435-445). Association for Computing Machinery. https://doi.org/10.1145/2568225.2568271

International Organization for Standardization, & International Electrotechnical Commission. (2023). *Systems and software engineering - Systems and software Quality Requirements and Evaluation (SQuaRE) - Product quality model* (ISO/IEC Standard No. 25010:2023).

International Telecommunication Union Radiocommunication Sector. (2011). *Studio encoding parameters of digital television for standard 4:3 and wide-screen 16:9 aspect ratios* (Recommendation ITU-R BT.601-7).

International Telecommunication Union Radiocommunication Sector. (2015). *Parameter values for the HDTV standards for production and international programme exchange* (Recommendation ITU-R BT.709-6).

Kalibera, T., & Jones, R. E. (2013). Rigorous benchmarking in reasonable time. In *Proceedings of the 2013 International Symposium on Memory Management* (pp. 63-74). Association for Computing Machinery. https://doi.org/10.1145/2464157.2464160

Kurtz, M., Murphy, E., & Kilshaw, C. (2025). Integrated quality control: Improving production efficiency to achieve scale at DNEG Animation. In *Proceedings of the Digital Production Symposium* (Article 10, pp. 1-4). Association for Computing Machinery. https://doi.org/10.1145/3744199.3744629

OpenCV contributors. (2025). *Image file reading and writing*. https://docs.opencv.org/4.12.0/d4/da8/group__imgcodecs.html

OpenImageIO contributors. (n.d.-a). *Cached images*. https://openimageio.readthedocs.io/en/stable/imagecache.html

OpenImageIO contributors. (n.d.-b). *Comparing images with idiff*. https://openimageio.readthedocs.io/en/stable/idiff.html

OpenImageIO contributors. (n.d.-c). *oiiotool: The OIIO Swiss Army Knife*. https://openimageio.readthedocs.io/en/stable/oiiotool.html

Pearsall, C., Hardie, A., Sakr, M., & Bigda, J. (2026, July 18). *Automating the handoff: From hardcoded QC scripts to a multi-agent architecture for layout-to-animation reviews* [Conference presentation]. Digital Production Symposium 2026, Denver, CO, United States. https://digiproconf.org/program/

Peffers, K., Tuunanen, T., Rothenberger, M. A., & Chatterjee, S. (2007). A design science research methodology for information systems research. *Journal of Management Information Systems, 24*(3), 45-77. https://doi.org/10.2753/MIS0742-1222240302

Qt Project. (n.d.). *QThread - Qt for Python*. https://doc.qt.io/qtforpython-6/PySide6/QtCore/QThread.html

Visual Effects Society Technology Committee. (2026). *VFX Reference Platform*. https://vfxplatform.com/

Wohlin, C., Runeson, P., Höst, M., Ohlsson, M. C., Regnell, B., & Wesslén, A. (2024). *Experimentation in software engineering* (2nd ed.). Springer. https://doi.org/10.1007/978-3-662-69306-3

Wright, S. (2024). *Digital compositing for film and video: Production workflows and techniques* (5th ed.). Routledge.

Appendix A. Repeatability and Reproduction Commands

The following commands repeat the main automated evidence from the project root. They are included as operational reference and as support for future independent replication, rather than as implementation code.

Source code and project repository: https://github.com/Joshmedinag/NCCA-mastersproject

Environment and tests

```
uv sync --python 3.12 --extra dev
```

```
uv run pytest --cov-fail-under=90
```

Structure inspection and analysis

```
uv run aovguard inspect-structure <path-to-frame.exr>
```

```
uv run aovguard analyze <path-to-source> --json report.json --html report.html
```

Sequence checking and report comparison

```
uv run aovguard check-sequence <path-to-render-folder>
```

```
uv run aovguard compare-reports baseline.json candidate.json --json comparison.json
```

Benchmark

```
uv run python experiments/benchmark_analysis.py
```

Appendix B. Recorded Benchmark Data

Field	AOV-first	Frame-first
Frames	12	12
AOVs	5	5
Resolution	320 x 180	320 x 180
Repetitions	30	30
Application reader calls	60	12
Median elapsed time	0.438742 s	0.118082 s
Interquartile range	0.072988 s	0.036943 s
Minimum elapsed time	0.257141 s	0.061700 s
Median Python-tracked peak memory	5,554,772 bytes	11,077,580 bytes
Pixel checksum	5,365,412.494333796	5,365,412.494333796

Each strategy received one unrecorded warm-up. The recorded execution order alternated between strategies on every repetition.

Appendix C. Final Verification Snapshot

Final local verification date: 14 August 2026.

Platform: Windows 11.

Python: 3.12.12.

Tests: 229 passed, 0 failed.

Total combined statement/branch coverage: 95.24%.

Coverage gate: 90% minimum.

Two external Maya/Arnold single-part multilayer files were processed successfully. The inspected example was 1920 x 1080 with 25 channels and eight inferred AOVs. One contextual empty-emission warning was produced.

Additional sequence workflow validation: 61 Maya/Arnold EXRs from frames 0000-0060 were processed successfully with zero failed reads. The inspected sequence contained 51 channels and 18 inferred AOV descriptors; sequence discovery reported four-digit padding, no missing frames and no duplicates.

Appendix D. Data and Evaluation Notes

No EXR source files are distributed with the repository or release archive. The `examples` directory is intentionally empty except for a placeholder file. Real renders should be selected from an authorised external location. Anonymised JSON and HTML report examples preserve the report structure without exposing local production paths or image data.

The formative usability protocol is documented in `docs/usability_evaluation.md`. Participant results must not be added unless the relevant university ethics route has been confirmed and the evaluation has actually been completed.