



National Centre for
Computer Animation

DCC Rendering Pipeline Tool

By Radeke K. Onamusi

MSc. Computer Animation and Visual Effects
National Centre for Computer Animation and Visual Effects
Bournemouth University
August 2025

ABSTRACT

Digital Content Creation (DCC) tools such as Autodesk Maya and SideFX Houdini form the backbone of modern visual effects and animation pipelines. While each application provides unique functionality, they share common requirements managing assets and shots, configuring renders, exchanging scene data, and preparing outputs for compositing. Traditionally, studios develop bespoke in-house pipeline solutions to address these needs, but such systems are often expensive and inaccessible to smaller teams or independent projects.

The program developed in this project operates seamlessly across **Python, Mayapy, and Hython** contexts, with subprocess management and custom command wrappers providing accessible entry points for both technical directors and artists.

By unifying workflows across applications, the tool reduces redundancy, enforces consistency, and improves interoperability. The results demonstrate that scalable, production-grade practices such as USD exchange, and renderer-agnostic look development can be achieved without the overhead of large proprietary systems, offering a lightweight yet effective solution adaptable to diverse VFX pipelines.

DEDICATION

Dedicated to my ever supportive husband and my entire family

ACKNOWLEDGEMENT

I am grateful to Jon Macey and the entire NCCA Staff who were always happy to help

ABSTRACT.....	2
DEDICATION.....	3
ACKNOWLEDGEMENT.....	4
1. Introduction.....	7
1.1 Motivation.....	7
1.2 Aims and Objectives.....	8
1.3 Project Scope.....	9
2. Related Work.....	9
2.1 Deadline Cloud.....	9
2.2 OpenCue.....	11
2.3 Project Approach.....	13
3. System Design and Architecture.....	13
3.1 Architectural Overview.....	14
3.2 Shots and Asset Management Module.....	14
3.3 Rendering Management Module.....	15
3.4 Post-Rendering Integration Module.....	16
3.5 Testing and Mocking Module.....	17
3.6 Technical Development and Libraries.....	18
3.7 System Interaction Flow.....	19
5. Conclusion and Future Work.....	23
5.1 Conclusion.....	23
5.2 Future Work.....	23
5.3 Closing Remarks.....	25

1. Introduction

1.1 Motivation

Digital Content Creation (DCC) tools such as Autodesk Maya and SideFX Houdini are the foundation of most VFX and Animation Pipelines. These applications share several common needs, despite their differences: they must manage assets, exchange data with other applications, configure render settings and deliver outputs to downstream departments such as compositing. Most DCC tools have a Python API that exposes core functionality, enabling technical artists and pipeline developers to script workflows, automate repetitive tasks, and integrate software into larger production ecosystems. There is often significant overlap in the workflows that must be developed and maintained across different DCC tools due to their similar functions and when implemented independently in each environment, these solutions can become redundant, inconsistent, and difficult to maintain.

	Maya	Houdini
Asset & Shot Management	Asset references via Maya scene files (.ma/.mb); manual versioning often required	Asset management via HIP files; versioning typically user-driven
Scene Export	Exports to Alembic, USD, animExport or native Maya formats	Exports to USD, Alembic, or bgeo
Rendering Configuration	Arnold with support for other 3rd Party renderers e.g Renderman, VRay	Karma and Matra with support for third party renderers such as Arnold
Shading/Lookdev	Arnold shaders, MaterialX support (via USD export)	VOP node equivalents of the MaterialX shader nodes.
Output for Compositing	Render outputs (EXR, AOVs) sent downstream to comp	Render outputs (EXR, AOVs) sent downstream to comp
Python API Integration	Mayapy	Hython

Table 1.1: Overlapping pipeline requirements in Maya vs. Houdini.

1.2 Aims and Objectives

The primary aim of this project is to design and implement a rendering pipeline tool that breaks down common production tasks into reusable modules that can run in different environments. By doing so, the project seeks to reduce redundancy, enforce consistency, and improve interoperability between Digital Content Creation (DCC) tools such as Autodesk Maya and SideFX Houdini.

This overarching aim can be broken down into the following specific objectives:

- Define and enforce standardized naming conventions for assets, shots, and outputs.
- Implement version tracking to ensure reproducibility and reduce the risk of overwriting.
- Use YAML-based project configuration files for lightweight, portable management of metadata.
- Unify rendering workflows across DCC tools
- Enable conversion of scenes into the Universal Scene Description (USD) format for cross-application compatibility.
- Apply MaterialX shading definitions to achieve consistent and realistic look development.
- Expose renderer configuration settings into a unified interface that translates to Arnold and Karma.
- Validate render outputs (resolution, frame range, and channel naming) for consistency across renderers.
- Demonstrate cross-renderer comparison between Arnold and Karma outputs.

Through these aims and objectives, the project demonstrates how a modular, Python-based tool can provide a unified rendering pipeline that scales across production environments while maintaining reliability, reproducibility, and flexibility.

1.3 Project Scope

This project simplifies and abstracts some of the common functions of DCC tools into a unified Python-based pipeline tool. It streamlines the process of maintaining separate workflows within each DCC tool, and provides a suite of modules that operate within a controlled virtual environment. The tool is designed to be deployed across VFX pipelines, enforcing consistency while still integrating with renderer- and DCC-specific features.

The key features of the tool are:

- **Shots and Asset Management** — Centralized naming conventions, version tracking, and configuration stored in lightweight YAML files, ensuring reproducibility and portability.
- **Rendering Management** — Conversion of scene data into USD format, application of MaterialX shading for renderer-agnostic look development, and management of rendering parameters across Arnold and Karma.

The environments supported are **Mayapy**, used for asset export and Arnold rendering, and **Hython**, used for USD import in Solaris and Karma rendering. Together, these environments demonstrate the tool's interoperability across major industry-standard DCC applications.

2. Background

In large-scale visual effects and animation production teams, proprietary pipeline tools are commonly developed to streamline workflows and ensure consistency across departments. These tools integrate multiple Digital Content Creation (DCC) applications and provide centralized control over the processes, reduce manual errors, improve reproducibility, and allow studios to scale production efficiently.

2.1 Deadline Cloud

Deadline Cloud, developed by Thinkbox (an Autodesk company), is a widely adopted render management solution in the industry. It allows studios to submit, manage, and monitor rendering jobs across large render farms or cloud-based resources. Deadline supports multiple DCC applications including Maya, Houdini, and Nuke, enabling artists to submit jobs directly from their preferred tools. Key features include distributed job scheduling, priority management, error handling, and cloud integration for scaling render resources dynamically.

By providing a centralized interface for render management, Deadline Cloud ensures that large teams can coordinate complex production tasks while maintaining asset integrity and reproducible outputs.

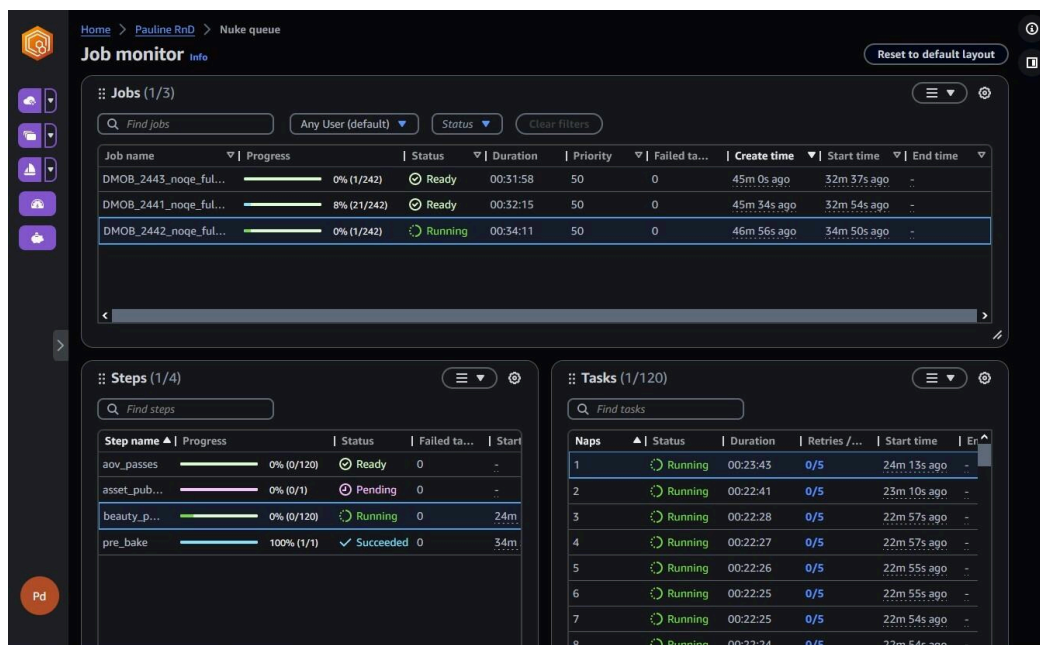


Fig 2.1 — Deadline Cloud Job Monitoring Interface.

Source: <https://aws.amazon.com/deadline-cloud/>

The success of Deadline Cloud is brought about by a set of functional requirements that define its operation within large-scale productions:

1. **Cross-DCC Integration:** Jobs can be submitted from multiple software applications, including Maya, Houdini, Nuke, and 3ds Max.
2. **Job Distribution and Monitoring:** The system can distribute tasks across multiple nodes, monitor progress in real time, and handle failures gracefully.
3. **Asset and Version Management:** Ensures that the correct versions of assets and scene files are used for each render, preventing inconsistencies and reproducibility issues.
4. **Cloud-Based Deployment:** Provides the ability to scale resources dynamically using cloud infrastructure, supporting large teams and fluctuating workloads.
5. **Cost Management:** Deadline Cloud provides an interface for tracking costs and allows users set up budgets

2.2 OpenCue

OpenCue is an open-source render management system designed to schedule, track, and distribute rendering tasks across a render farm. Originally developed by Sony Pictures Imageworks, it has been adopted by multiple studios due to its flexibility, scalability, and integration capabilities.

OpenCue is built on a client-server model:

Cuebot (Server): Core manager that maintains the render job queue, monitors render hosts, and distributes tasks.

Cuegui (Client GUI)/ Cueweb: Provides artists and technical directors with an interface to submit jobs, monitor progress, and manage priorities.

Render Hosts (Workers): Machines that execute rendering tasks assigned by Cuebot. These can be individual workstations or nodes in a render farm.

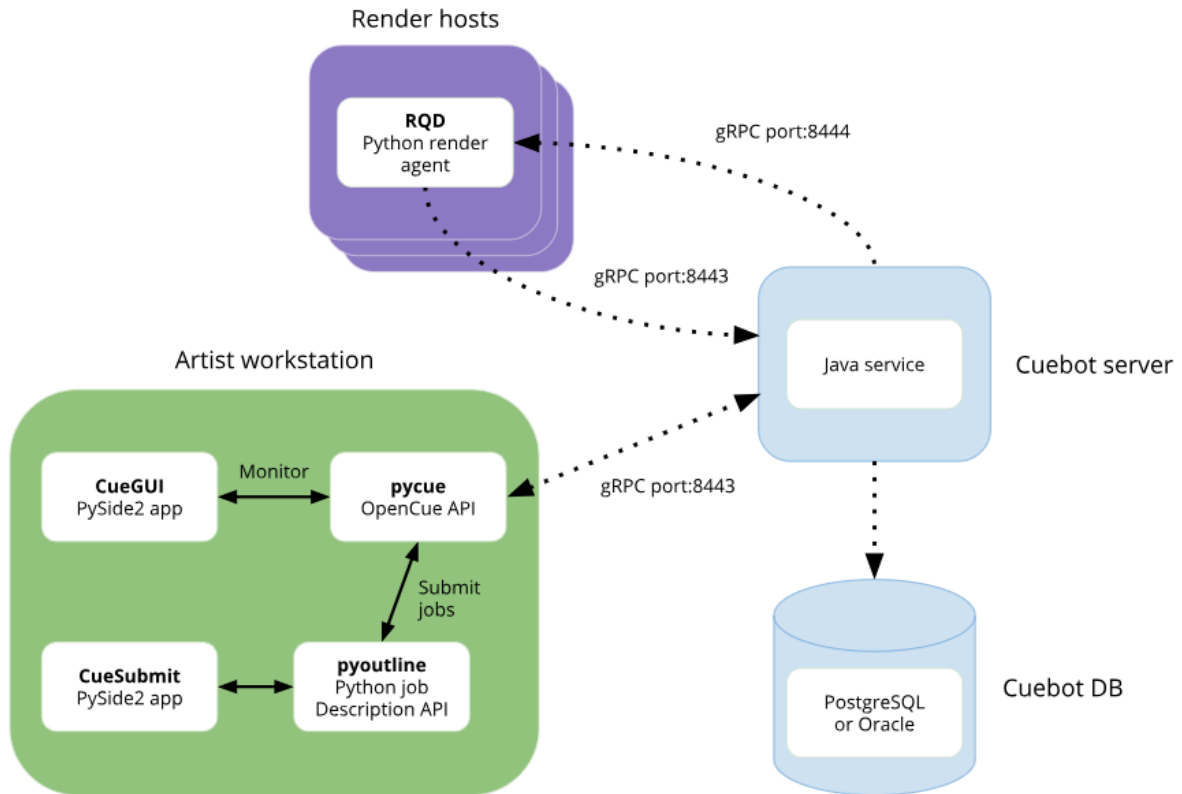


Fig 2.2 – How the various components interact in a large-scale deployment of OpenCue

Source: <https://docs.opencue.io/docs/concepts/opencue-overview/>

Key Features of OpenCue include;

Job Management: Supports submission of multi-frame jobs with dependencies, priorities, and resource requirements.

Monitoring and Reporting: Real-time tracking of job progress, completion status, and host performance.

Cross-Platform Support: OpenCue supports multiple render engines and can integrate with Maya, Houdini, Blender, Arnold, RenderMan, and more.

Extensibility: Being open-source, studios can customize the system to fit pipeline-specific requirements.

Unlike Deadline Cloud, OpenCue is free to use.

2.3 Project Approach

The pipeline tool presented in this thesis is designed for lightweight deployment, small teams, or academic projects. It addresses challenges such as asset management, rendering configuration, and cross-DCC interoperability — while remaining modular and accessible.

The key features of this project include:

- **Cross-DCC Integration:** The tool supports Mayapy for Maya and Hython for Houdini, enabling USD-based scene exchanges and renderer-agnostic shading workflows.
- **Asset and Shot Management:** Naming conventions, version tracking, and project configuration are centralized in YAML files to ensure consistency and reproducibility.
- **Rendering Management:** Supports USD conversion, application of MaterialX shading, and configuration of both Arnold and Karma renderers.

3. System Design and Architecture

The pipeline tool is designed to streamline the management of 3D animation and visual effects projects. It consists of a graphical user interface (GUI), a project configuration manager, and a render management module. The architecture emphasizes modularity, enabling independent testing of components, extensibility for multiple render engines, and robustness for handling large project datasets

By carrying out workflows as discrete components, the system allows artists and technical directors to interact with Maya and Houdini through a consistent Python interface while maintaining renderer-agnostic behavior. The architecture prioritizes ease of configuration, and testability, ensuring that each module can be maintained independently and integrated seamlessly into larger pipelines.

3.1 Architectural Overview

At a high level, the system consists of four core modules:

1. **Shots and Asset Management:** Handles naming conventions, version tracking, and configuration management.
2. **Rendering Management:** Manages USD file conversion, MaterialX shading application, and renderer-specific settings.
3. **Graphical User Interface:** An easy-to-use interface to access the functionalities of the program in minimal time
4. **Testing and Mocking:** Ensures stability and reliability through automated unit and integration tests.

These modules are connected via a central Python environment, which exposes high-level interfaces for each workflow while working within the underlying DCC-specific environments and APIs. Both Mayapy and Hython act as execution environments, allowing the tool to operate directly inside each DCC tool which was achieved through subprocess management.

3.2 Shots and Asset Management Module

This module is responsible for organizing and maintaining project data in a reproducible and consistent manner. Its key features include:

- **Naming Conventions:** Enforces standardized names for assets, shots, and versions to prevent ambiguity across departments.
- **Version Tracking:** Automatically tracks scene versions, asset iterations, and publishes metadata.
- **Project Configuration:** Uses lightweight YAML files to define project-specific parameters such as asset paths, render settings, and environment variables.

```
renders:
  rsv001:
    frames:
      - 1
      - 2
      - 3
    settings:
      camera: /camera1
      denoise: false
      filename_template: '{shot}_{camera}_{frame}'
      fps: 24
      light: Dome Light
      motion_blur: false
      output_dir: TEMP/Gallery/Renders
      output_format: PNG
      renderer: Karma
      resolution_height: 1080
      resolution_width: 1920
  rsv002:
    frames:
      - 1
      - 2
      - 3
    settings:
      camera: /camera1
      denoise: false
      filename_template: '{shot}_{camera}_{frame}'
      fps: 24
      light: Dome Light
      motion_blur: false
      output_dir: TEMP/Gallery/Renders
      output_format: PNG
      renderer: Arnold
      resolution_height: 1080
```

Fig 3.1 Snapshot from Project- YAML Configuration

The module provides both Python API calls and command-line interfaces to interact with project data, allowing users to access, modify and interact with project assets without being required to leave their DCC environment.

3.3 Rendering Management Module

The rendering module is designed to be **renderer-agnostic** while accommodating DCC-specific requirements. Key responsibilities include:

- **USD Conversion:** The program converts Maya scene files into USD format when the user chooses to use Karma, thereby achieving interoperability between Maya and Houdini.
- **MaterialX Shading:** MaterialX-based shaders were applied to ensure consistent look development across different render engines. While MaterialX provides a standard for physically plausible materials, some adaptation was necessary to map Maya's MaterialX shader nodes to equivalent or near-equivalent features in Houdini/Karma.
- **Renderer Configuration:** The program handles renderer-specific parameters for Arnold (via Mayapy) and Karma (via Hython), including sampling, lighting, and output settings.

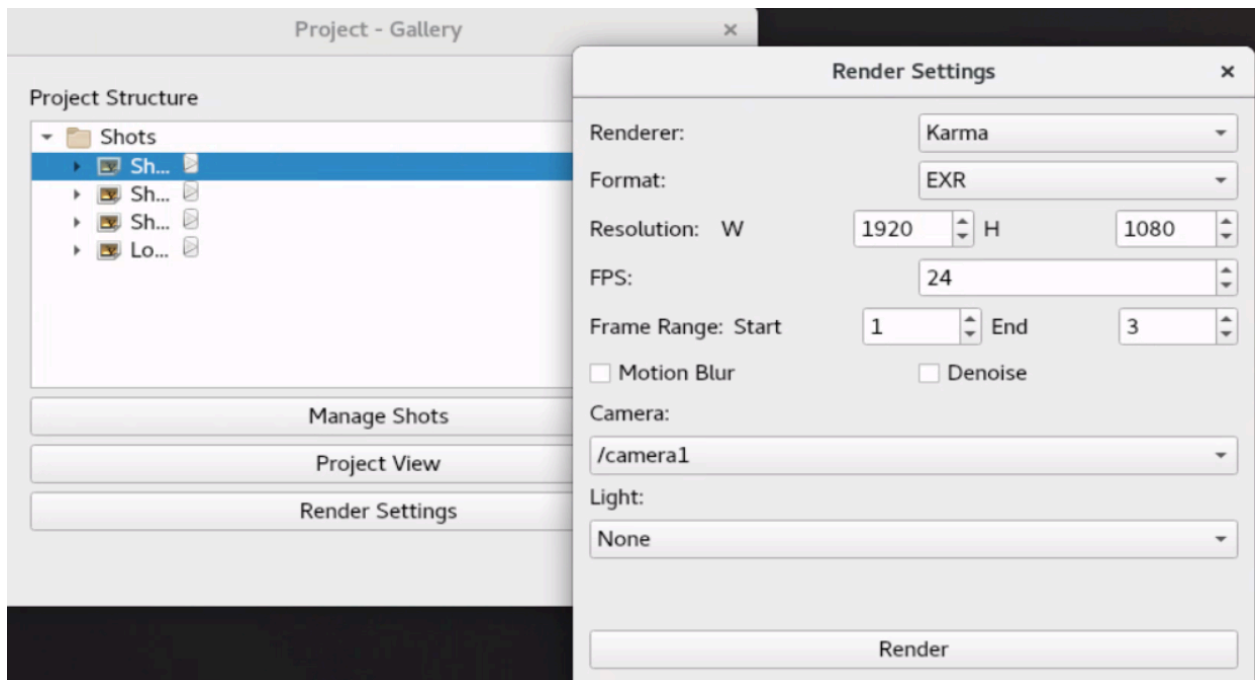


Fig 3.2- Render setting window from Rendering Pipeline Tool

3.4 Post-Rendering Integration Module

While not a full-featured compositing application, this module ensures that render outputs meet minimum quality and consistency standards before they are passed to the compositing team. The module inspects the RGB channels of the rendered output

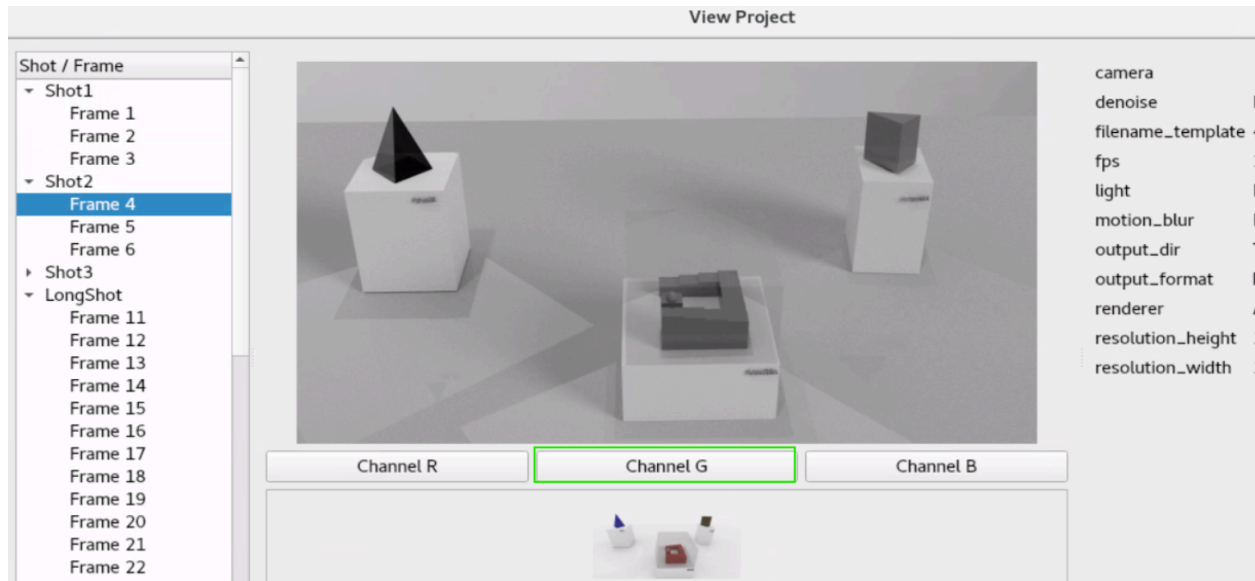


Fig 3.4 — Example of a render verification workflow, showing image green channel.

The module also allows for users to inspect render settings, compare versions and select best output easily.

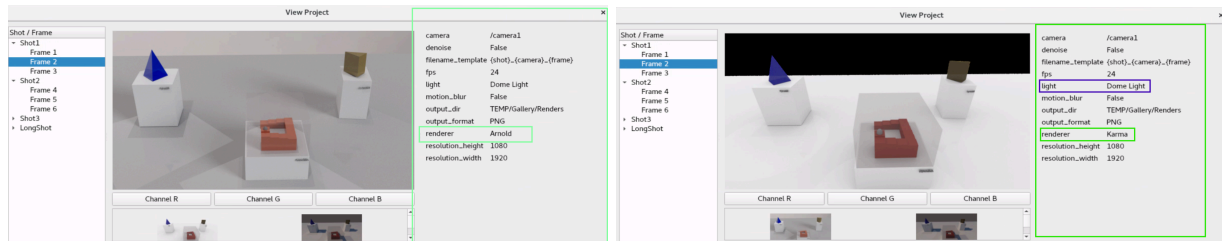


Fig 3.5, Fig3.6 — Example of a render verification workflow, showing render output comparison.

3.5 Testing and Mocking Module

Reliability is a critical consideration in pipeline tools. The testing module provides:

- **Unit Tests:** Using **pytest** to verify the correctness of individual functions and classes.
- **Mocking DCC APIs:** Using **unittest.mock** to simulate Maya and Houdini behavior during testing, allowing tests to run outside of the DCC environment.
- **Integration Tests:** Ensuring that the full workflow — from asset creation to render output — behaves as expected.

This module ensured that changes in one part of the pipeline does not cause breakage to other parts, and ensured that potential faults were caught and addressed early .

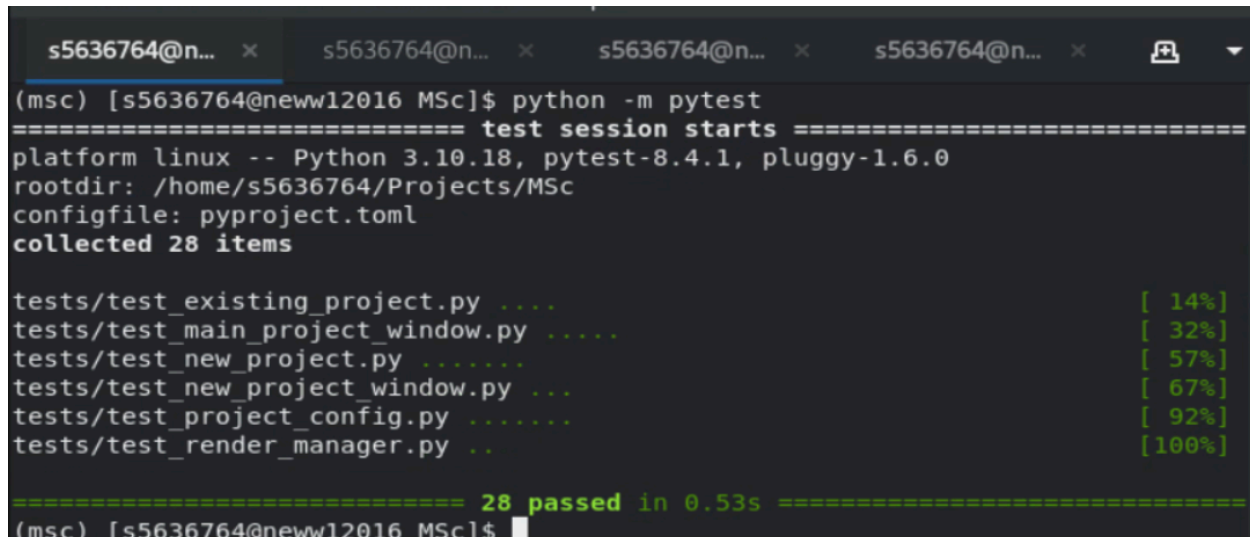
A terminal window with a dark background and light-colored text. The prompt is '(msc) [s5636764@neww12016 MSc]\$'. The command 'python -m pytest' has been executed. The output shows 'test session starts' followed by platform and version information: 'platform linux -- Python 3.10.18, pytest-8.4.1, pluggy-1.6.0'. The root directory is '/home/s5636764/Projects/MSc' and the config file is 'pyproject.toml'. It then says 'collected 28 items'. A progress bar shows the progress of test files: 'tests/test_existing_project.py [14%]', 'tests/test_main_project_window.py [32%]', 'tests/test_new_project.py [57%]', 'tests/test_new_project_window.py ... [67%]', 'tests/test_project_config.py [92%]', and 'tests/test_render_manager.py .. [100%]'. At the bottom, it says '==== 28 passed in 0.53s =====' and the prompt returns to '(msc) [s5636764@neww12016 MSc]\$'.

Fig3.7— All tests passed

3.6 Technical Development and Libraries

The development of the system relied on a set of well-established libraries and frameworks to ensure maintainability, cross-platform compatibility, and interoperability with existing VFX tools. Each component was chosen for a specific role in supporting the design goals of modularity, extensibility, and adherence to industry standards.

User Interface (PySide2 / Qt for Python):

The front-end of the system was implemented using PySide2, the official Python bindings for the Qt framework. This enabled the creation of a modern, platform-independent interface consistent with DCC (Digital Content Creation) application standards. Qt's vast widget set and event-driven model were leveraged for project management dialogs, shot selection, and render submission workflows.

Configuration Management (YAML):

Project metadata, including scene paths, shot definitions, and render settings, was serialized using the YAML format. YAML was chosen for its readability, hierarchical structure, and native support in Python through the **PyYAML** library. This allowed configuration files to serve both as machine-readable state for automation and human-readable documents for artists or pipeline engineers. The ProjectConfig class managed load/save operations, ensuring consistency across project sessions.

Image I/O (OpenImageIO):

OpenImageIO (OIIO) was employed for image sequence inspection and metadata handling. OIIO's extensive support for industry-standard image formats (EXR, TIFF, JPEG, PNG, etc.) and its efficient handling of high-dynamic-range (HDR) data made it an ideal choice for previewing renders and verifying output consistency across rendering engines. This library also provided access to technical metadata such as resolution, channel information, and frame range, supporting automated validation tasks in the pipeline.

Scene and Render Interchange (USD + MaterialX):

The system integrated Pixar's USD (Universal Scene Description) as the core scene interchange format, facilitating asset transfer between Maya and Houdini/Solaris. USD's composition model allowed the system to preserve hierarchy, references, and variations. On top of USD, MaterialX was adopted to standardize shader networks. The project took advantage of MaterialX's support for physically-based shading models, color management, and layered materials as highlighted by (Harrysson, Smythe & Stone, 2021). Some technical development was required to bridge gaps between Maya's MaterialX implementation and Houdini/Karma's equivalent nodes, to ensure shading consistency.

Automation and Scripting (Python):

The choice of Python as the primary language was motivated by its widespread adoption in VFX pipelines, strong ecosystem of libraries, and interoperability with both Maya (MayaPy) and

Houdini (Hython). Python's flexibility allowed the system to unify environment-specific scripts into a coherent cross-application workflow.

3.7 System Interaction Flow

The typical workflow starts with the **asset and shot management module**, which prepares project data. Scene files are then exported and converted to USD, and MaterialX shaders are applied. The rendering module submits jobs to the appropriate renderer (Arnold or Karma). Finally, the post-render module validates outputs, and the testing module ensures that the entire workflow executes correctly.

4. Environments and Execution Contexts

The pipeline tool is designed to operate within multiple Digital Content Creation (DCC) environments, with a focus on Autodesk Maya (Mayapy) and SideFX Houdini (Hython). These contexts were chosen because they represent widely adopted industry-standard tools, each with Python APIs that allow deep integration into production workflows.

The project architecture treats these environments not as separate silos but as interchangeable execution contexts, ensuring that workflows can be transferred with minimal modification. This section outlines how the tool operates inside Mayapy and Hython, and how it leverages Python's subprocess module and custom commands for process management.

4.1 Working Within Python

At its core, the tool is written in standard Python 3.10, ensuring that modules can be executed outside of any DCC. This “vanilla” Python mode is essential for:

- **Development and Testing:** Core functionality such as configuration parsing, version tracking, and USD file handling can be validated independently of a DCC.
- **Continuous Integration:** Automated tests can run in lightweight environments without requiring Maya or Houdini installations.
- **Command-line Utilities:** Artists and technical directors can access features (e.g., asset version queries or batch renders) via CLI commands that wrap the same Python APIs used inside DCCs.

By maintaining a strict separation between “core logic” and “DCC bindings,” the system ensures portability and modularity.

4.2 Mayapy

Mayapy is Autodesk Maya's standalone Python interpreter. It provides full access to Maya's Python API (`maya.cmds`, `maya.api.OpenMaya`) without requiring the graphical interface to be launched.

Within this project, Mayapy is primarily used for:

- **Scene Export:** Preparing assets and shots for downstream rendering.
- **Arnold Rendering:** Submitting renders directly through Maya's integrated Arnold renderer.
- **Data Conversion:** Exporting scene data into USD for interoperability with Houdini.

The pipeline modules are invoked within Mayapy scripts to automate repetitive export and render tasks. This eliminates reliance on user-driven GUI interactions, ensuring reproducibility and efficiency.

4.3 Hython

Hython is SideFX Houdini's headless Python interpreter. Like Mayapy, it grants full access to Houdini's API (hou module), including Solaris, Houdini's USD-based layout and rendering context.

In this project, Hython is used for:

- USD Import: Loading USD assets exported from Maya into Solaris.
- Karma Rendering: Managing renderer settings and rendering through Houdini's Karma engine.
- Look Development Validation: Applying MaterialX shaders inside Houdini for consistent shading across renderers.

The ability to automate Solaris operations through Hython demonstrates the tool's renderer-agnostic philosophy, bridging Arnold (Maya) and Karma (Houdini).

4.4 Subprocess Management

In order to facilitate automation, the pipeline frequently needs to launch DCC-specific interpreters (Mayapy, Hython) from an external script or command-line interface. This is achieved using Python's **subprocess** module, which allows spawning new processes while capturing their outputs.

Typical use cases include:

- Batch Rendering: Launching multiple Mayapy or Hython processes to process assets in parallel.
- Isolated Testing: Running individual pipeline stages in sandboxed environments to prevent state contamination.
- Cross-DCC Execution: Invoking Maya-export scripts followed by Houdini-import scripts in a single automated pipeline.

By treating each DCC session as a subprocess, the system maintains fault isolation: if one stage fails, it does not compromise the rest of the workflow.

4.5 Custom Commands

To streamline user interaction, the tool defines custom command wrappers around common pipeline actions. These commands abstract the complexity of subprocess calls, exposing a simple interface to artists and technical directors.

An example present in this project is:

```
cmd = [  
    MAYAPY,  
    str(Path("adapters/maya_adapter.py").resolve()),  
    "render",  
    "--file", str(scene),  
    "--scene", self.metadata.get('project_name'),  
    "--output", str(out_file),  
    "--startf", str(frame),  
    "--endf", str(frame),  
    "--ext", extension  
] → Launches Mayapy rendering script.
```

5. Conclusion and Future Work

5.1 Conclusion

This thesis has presented the design and implementation of a **Python-based DCC rendering pipeline tool** aimed at unifying workflows across Autodesk Maya and SideFX Houdini. By abstracting common production needs into modular components, the system reduces redundancy, enforces consistency, and supports interoperability across environments.

The project demonstrated several key contributions:

- **Shots and Asset Management** through YAML-driven configuration, ensuring reproducibility and lightweight portability.
- **Rendering Management** that leverages USD and MaterialX to enable cross-application data exchange and renderer-agnostic look development.
- **Post Rendering Integration** with basic validation workflows, ensuring consistent outputs across renderers in downstream departments.
- **Testing and Mocking** using **pytest** and **unittest.mock** to establish reliability and simulate DCC API interactions.
- **Multi-environment execution** in Python, Mayapy, and Hython, with subprocess management and custom commands bridging tasks seamlessly.

Collectively, these contributions highlight how a relatively lightweight tool can achieve the type of functionality usually reserved for bespoke, in-house pipeline solutions developed by large studios. Importantly, this work shows that smaller teams and academic projects can adopt modern standards like USD and MaterialX while maintaining flexibility and scalability.

5.2 Future Work

While the project has successfully demonstrated its aims, several avenues remain open for future development:

1. **Graphical User Interface (GUI) Enhancements**
 - Expand the command-line wrappers into a cross-platform GUI (e.g., PySide2 or Qt for Python) for greater accessibility by artists.
 - Provide visual dashboards for version tracking, render submissions, and validation results.
2. **Deeper Compositing Integration**

- Extend beyond basic validation into automated Nuke or COPs script generation, linking renders directly into standardized comp templates.
- Enable shot-level review tools for quick visualization of outputs in editorial or dailies workflows.

3. **Render Farm and Cloud Integration**

- Connect the tool with render farm managers such as **Deadline** or **OpenCue** to handle job scheduling and scaling.
Explore cloud-native workflows that leverage Deadline Cloud or AWS Thinkbox services.

4. **Asset Dependency Management**

- Introduce graph-based tracking of assets and their dependencies (rigs, textures, caches), potentially using lightweight database backends (SQLite or PostgreSQL).
- Provide automatic invalidation or re-render triggers when upstream assets change.

5. **Performance and Parallelization**

- Implement asynchronous execution models for USD export and rendering tasks, taking advantage of multi-core environments.
Profile pipeline bottlenecks and optimize heavy subprocess operations.

6. **Broader DCC Support**

- Expand bindings to include Blender, Katana, or Unreal Engine, increasing interoperability across industries beyond VFX.
- Develop plugin-based architecture so new DCCs can be integrated with minimal effort.

References

Harrysson, N., Smythe, D. & Stone, J. (2021) *MaterialX Physically Based Shading Nodes, Version 1.38*. Autodesk, Industrial Light & Magic, Lucasfilm Advanced Development Group.[1]

Universal Scene Description

<https://openusd.org/release/index.html>

SideFX Houdini

<https://www.sidefx.com/>

Autodesk Maya,

<http://www.autodesk.com/products/autodesk-maya/>

OpenCue

<https://docs.opencue.io/>