VFX Pipeline an der HdM

https://gitlab.com/eldur_vfx/stupro_pipeline

Geschrieben von Katharina Böttcher

Denise Röhrich

Überarbeitet von Katharina Böttcher

Stand: 19.09.2022

Inhaltsverzeichnis

Einführung

Was heißt Pipeline überhaupt? Die Pipeline dient dazu Abläufe in der Postproduktion zu vereinfachen und zu automatisieren. Das beinhaltet verschiedene Themen:

- Umgebungsvariablen setzen
- Integration von Programmen (DCCs)
- Tools und Plug Ins für Programme schreiben
- Speicherortmanagement

Um eine Pipeline aufzusetzen, müssen verschiedene Fragen über das aktuelle Projekt geklärt werden:

- Wird ein Projektmanagement-Tool wie ShotGrid oder ftrack verwendet? Hier kann verwaltet werden, wer zu welchem Zeitpunkt an welchen Aufgaben sitzt. Es können Reviews auf Animationen oder Compositing gegeben werden und man erhält einen Überblick über den aktuellen Stand des Projekts.

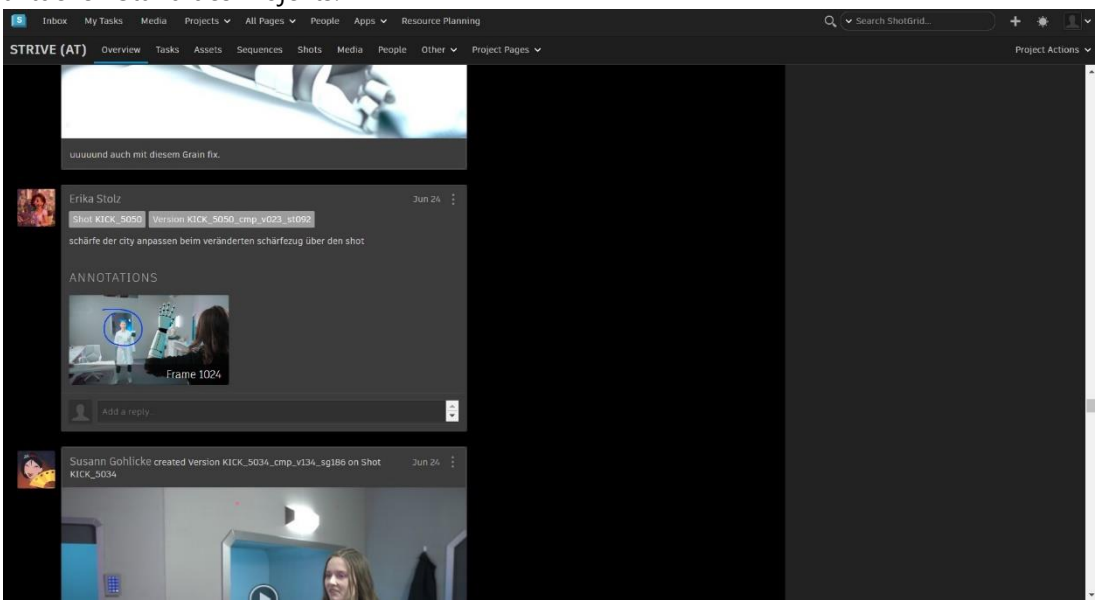

Abbildung 1: Übersicht in ShotGrid über die aktuellen Arbeiten an einem Projekt. Quelle: Eigene Abbildung.

- Wo sollen welche Dateien gespeichert werden? Wie sollen sie einheitlich benannt werden? Dies erleichtert allen Projektbeteiligten das Finden von Dateien und spart somit Zeit
- Welche Programme und welche Versionen der Programme sollen verwendet werden? Welche Render Engine dazu?
- In welchen Formaten erfolgt die Ausspielung von Shots?

Der Pipeline TD (Technical Director) ist dafür verantwortlich, dass diese Pipeline läuft und auch neue Funktionalitäten hinzugefügt werden. Insbesondere in der Integration mit ShotGrid ist auch die Kommunikation mit dem VFX Coordinator wichtig.

Pipeline Tools schreiben

Als Pipeline TD arbeitet man in aller Regeln mit der Programmiersprache Python. Im Vergleich zu Java weist sie eine deutlich einfachere Syntax auf. Einen guten Einstieg in die Programmierung mit Python bekommt man auf: https://www.w3schools.com/python/.

Alle gängigen DCCs haben in der Regel eine eigene Python API, mit der in Nuke beispielweise Nodes erstellt werden können. Diese APIs sind von den einzelnen DCCs gut dokumentiert und hier zu finden:

- Nuke:
  - Developer Guide: https://learn.foundry.com/nuke/developers/130/pythondevguide/
  - Alle Funktionen: https://learn.foundry.com/nuke/developers/120/pythonreference/
- Maya:
  - Developer Guide: https://knowledge.autodesk.com/support/maya/learn-explore/caas/CloudHelp/cloudhelp/2020/ENU/Maya-Scripting/files/GUID-55B63946-CDC9-42E5-9B6E-45EE45CFC7FC-htm.html
  - Alle Funktionen: https://help.autodesk.com/view/MAYAUL/2022/ENU/index.html?contextId=COMMANDSPYTHON-INDEX
- Houdini:
  - Developer Guide: https://www.sidefx.com/docs/houdini/hom/index.html
  - Alle Funktionen: https://www.sidefx.com/docs/houdini/hom/hou/index.html

Programme, welche nicht mit Python arbeiten, sondern mit Javascript sind beispielsweise Photoshop und After Effects.

Am einfachsten ist es, wenn man die Pipeline über PyCharm bearbeitet. Das ist eine Entwicklungsumgebung für Python, welche kostenfrei heruntergeladen werden kann.

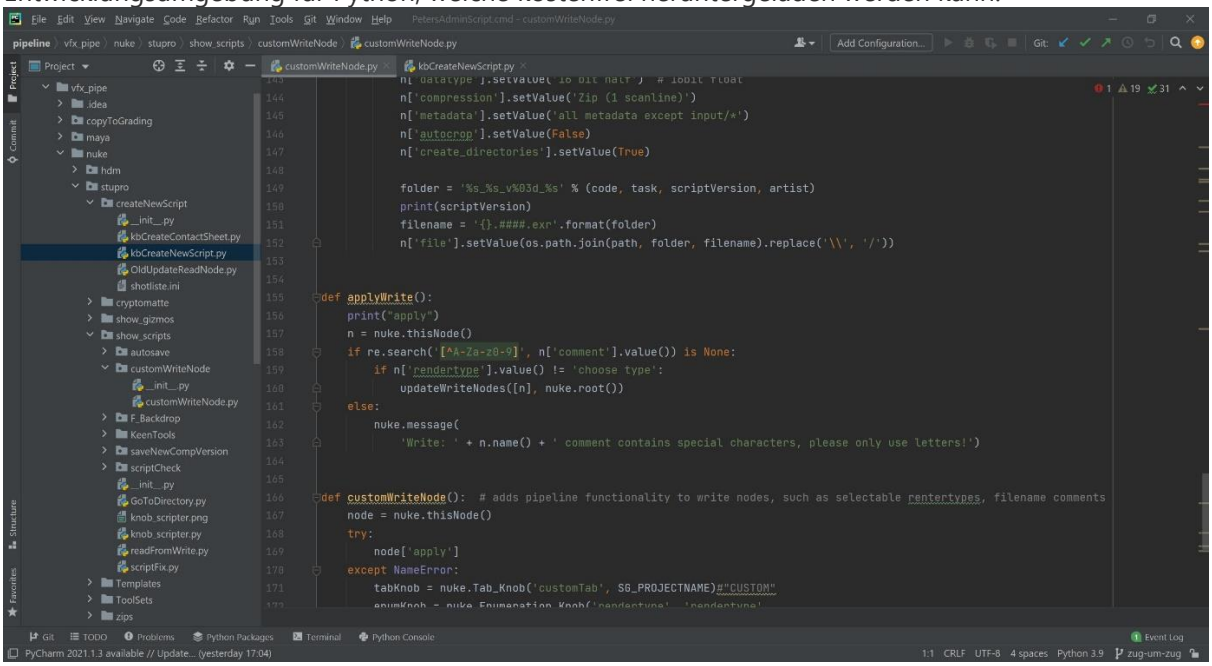

Abbildung 2: UI von PyCharm. Quelle: Eigene Abbildung.

How To: Eigenes PlugIn schreiben

Am einfachsten ist es in dem Programm selbst zu starten. Zunächst sollte man sich überlegen, was genau die Anforderungen an ein Programm sind und welche Informationen es dazu benötigt. Dann kann die Recherche beginnen, wie man diese Informationen herausfindet. Gefundene Code Zeilen können schnell im Script Editor des Programmes getestet und modifiziert werden. Allmählich erhält man so die notwenigen Zeilen für sein Plugin. Das Vorgehen ist dabei unabhängig davon, ob es sich um ein Script handelt, welches aus einem Menü heraus aufgerufen wird oder um ein Script, welches auf einer Node läuft. Funktioniert soweit alles im Script Editor kann der Code rauskopiert und in PyCharm in das bestehende Dateisystem integriert werden. Hier muss dann noch an den richtigen Stellen ergänzt werden, wie ein Programm das Plugin lädt. Dazu sucht man am besten Beispiele, wie andere Tools integriert wurden. Pipeline Scripting besteht an sich einfach aus Googlen, Code kopieren und anpassen und Debuggen!

How To: Debuggen

Debugging beschreibt das Finden von Fehlern, die in einem Programm auftreten. Dazu gibt es verschiedene Möglichkeiten, die je nach Umgebung besser oder schlechter funktionieren:

- Log Dateien: In Log Dateien werden Fehler geloggt, ohne dass der Developer dafür zunächst etwas tun muss. ShotGrid hat recht gute Logs, die häufig bereits weiterhelfen. Teilweise kann es hier helfen in ShotGrid Desktop zusätzlich das Debugging anzuschalten, um noch mehr Informationen zu erhalten.
- Print Commands: Es ist möglich in den Code print() Statements zu schreiben. Hier können dann Variablen in die Console geprintet werden. Somit kann herausgefunden werden, welche Variable einen falschen Wert beinhaltet und somit zum Fehler führt. Genauso kann man mit print Statements feststellen, an welchem Punkt eines Programms ein Fehler passiert, der dazu führt, dass das Programm abbricht. Dazu einfach alle paar Zeilen oder vor jedem Beginn von if oder for Statements eine fortlaufende Zahl printen. Die ersten Zahlen wird man vom Programm noch zurückhalten. Der Fehler passiert dann zwischen der letzten geprinteten Zahl und dem darauffolgenden print Statement.
- pdb: pdb ist ein Python Modul zum Debuggen. Mit

  ```
  import pdb
  pdb.set_trace()
  ```

  wird ein Breakpoint gesetzt. Das Verfahren funktioniert nur, wenn eine Console vorhanden ist, in der auch Befehle eingegeben werden können. Dies ist beispielsweise in Houdini der Fall. An dem Breakpoint wird dann in der Console gestoppt und die nächste auszuführende Zeile angezeigt. Mit dem Command c wird bis zum nächsten Breakpoint das Programm weiter ausgeführt, mit n wird nur die nächste Zeile ausgeführt. Ist das Programm gestoppt, kann man beliebige Python Funktionen ausführen.
- Programmzeilen einzeln testen: Das Verfahren funktioniert gut in eigentlich allen DCCs, abhängig davon wie komplex der Code ist. Dazu werden die benötigten Programmzeilen aus den Pythondateien in den Script Editors des DCCs kopiert. Achtung bei Nuke: Aus PyCharm kann man irgendwie nicht nach Nuke kopieren. Gegebenenfalls müssen beim Kopieren einzelne Zeilen angepasst werden, zum Beispiel wenn Umgebungsvariablen oder globale Variablen gesetzt werden. Im Anschluss kann man entweder den ganzen Code ausführen und Nuke zeigt an, wo ein Fehler auftritt oder einzelne Zeilen und dann auch den Inhalt von Variablen abfragen.
- PyCharm Debugger: PyCharm hat die Funktion, dass man einen Debugger an einen laufenden Prozess heften kann. In der PyCharm Umgebung kann man dann mit Klick auf eine Zeile einen Breakpoint setzen. Wird das Programm ausgeführt, so hält es dann an diesem Punkt. Dies funktioniert teilweise mit ShotGrid, teilweise aber auch nicht. Im folgenden Bild sieht man, wo

der Debugger gestartet wird. Danach kann ein Python Prozess ausgewählt werden, welcher überwacht werden soll.

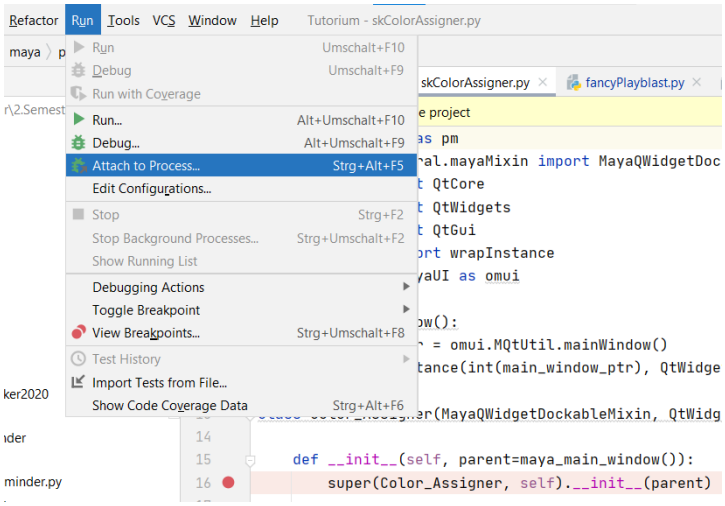

Abbildung 3: PyCharm Debugger starten. Quelle: Eigene Abbildung.

- PyDev und Eclipse: Diese Kombination funktioniert meist in mehr Fällen als der PyCharm Debugger. Bevor gestartet werden kann, muss in Eclipse PyDev installiert werden. Die Dokumentation dazu findet man hier: https://www.pydev.org/manual_101_install.html. Wechselt man die Ansicht von Eclipse dann in den Debug Mode, so sollte oben in der Menuleiste ein Reiter Pydev auftauchen. In der Symbolleiste unter drunter, gibt es die Möglichkeit den Pydev Server zu starten. Um einen Breakpoint zu setzen, muss folgende Zeile in den Code, an der gewünschen Stelle, hinzugefügt werden:

```
import pydevd, pydevd.settrace('localhost', port=5678, stdoutToServer=True, stderrToServer=True)
```

Gegebenenfalls muss die Nummer des Ports angepasst werden, diese steht in Eclipse in der Console, nachdem der Pydev Server gestartet wurde. Bei einer Ausführung des Programmes, egal in welcher Umgebung, sollte Eclipse dann an der richtigen Stelle stoppen. Ausnahmen bestätigen hier leider die Regel.

How To: GUI bauen in Python

Insbesondere bei den Maya Plugins gibt es in der Pipeline bereits Tools, welche ein Fenster öffnen und für die diese GUI (Graphical User Interface) von einem Pipeline TD geschrieben wurde. Hier bietet sich für Python Qt an. Auch Shotgun und andere Programme sind Qt basiert. Qt ist ein Modul zur Erstellung für User Interfaces ursprünglich in C++, welches mit PySide2 bzw. PyQt5, aber auch für Python zugänglich gemacht wurde.

Am besten probiert man selbst mal aus einfache UIs zu erstellen und nimmt sich ein Beispiel beispielsweise an dem Aufbau vom RenderChecker. In der Regel wird zunächst ein Fenster erstellt. Dann benötigt man ein Layout auf welchem verschiedene Widgets, wie Buttons oder Textfelder hinzugefügt werden können. Die Buttons können dann über connect (button.clicked.connect(customFunction)) mit einer beliebigen Funktion verknüpft werden, welche auf Knopfdruck ausgeführt werden soll. Dadurch bekommt das UI seine Funktionalität.

Die Dokumentation oder auch StackOverflow helfen einem hier gut weiter. Teilweise findet man eher Antworten, die in C++ geschrieben sind. Diese sind in der Regel auf Python anwendbar. Gut zu wissen ist hier, dass eine Funktion auf einem Objekt nicht mit einem Punkt aufgerufen wird, sondern mit einem Pfeil (statt button.setEnabled ist es dann button->setEnabled). Desweiteren wird in C++ überall der Objekttyp dazu geschrieben, was in Python überflüssig ist.

- ➔ Getting Started: https://doc.qt.io/qt-6/gettingstarted.html
- ➔ Documentation: https://doc.qt.io/all-topics.html

Starten eines Projekts

Vorgehen Lokal

1. Wiki lesen
2. PC einrichten mit PyCharm
3. ShotGrid einrichten:
    - Neues Projekt erstellen in Shotgun, gegebenenfalls die Voreinstellungen vom letzten Projekt übernehmen
    - Das Team erstellen (noch auf Inaktiv lassen) und dem Projekt hinzufügen
    - Shots, Assets und Tasks erstellen, sobald ein Breakdown vorhanden ist
        - WICHTIG: Benennungsstyle beachten. In der Regel werden vier Buchstaben für die Sequenz verwendet und im Anschluss die Shotnummer aus dem Breakdown.
        - Assets über die Seite Assets/Add Asset: Name eingeben, dann Type auswählen und dementsprechend das Template wählen. Durch die Wahl eines Templates werden automatisch Tasks für die Aufgabe gesetzt.
        - Shots über die Seite Shots/Add Shot dann einen ShotCode erstellen, dann ein Task Template auswählen und den Shot einer Sequence (Sequenzen werden aus dem Drehbuch abgeleitet, z.B. gehören alle Shots, die hinter einander an einem Ort spielen zu einer Sequenz.) zuordnen. Nach der Erstellung kann der VfX Supervisor noch dazu schreiben, ob der Shot eine Kamerabewegung hat und welche focalLength (Brennweite) der Shot hat - das ist dann für die Artist später eine gute Nachschlagquelle.
        - Tasks werden durch Auswahl der Templates automatisch erstellt, hier muss dann noch der Artist, der Reviewer und Start und Enddatum eingetragen werden und falls Tasks nicht gebraucht werden, können diese gelöscht werden (Task anhaken und über More/Send Selected to Trash)
    - Software Pfade auf Shotgun aktualisieren, sodass man die Programme, die auch auf den PCs sind, über Shotgun öffnen kann
4. Pipeline einrichten:
    - Dazu müssen die Gruppenvariablen angepasst werden, damit sie auf den aktuellen StuPro Ordner zeigt. Dies macht in der Regel Peter. Man kann die Variablen prüfen, indem man über cmd ins Command Fenster SET eingibt. Dann werden alle gesetzten Umgebungsvariablen gelistet.
    - Umbenennung von Variablen in der PIPELINE_CONSTANTS.json Datei im vfx_pipe Ordner. In den Variablen ProjectID und Name müssen die aktuellen StuPro Daten stehen. Hieraus holt sich die Pipeline die Benennungen für z.B. die Menus in Nuke und Maya, so dass diese immer wie die StuPro selbst heißen.
    - ShotGrid Pipeline Konfiguration erstellen. Mehr Informationen dazu stehen im ShotGrid Kapitel. Im Anschluss überprüfen, dass Nuke/Maya den StuPro und Shotgun Tab im Menu haben.
5. Farbraum anpassen, ob man die ACES oder Filmlight OCIO Konfiguration verwenden möchte. Das muss in einigen Python Dateien angepasst werden. Da einfach die Scripte einmal durchgehen und überprüfen (ggf. Notepad++ und alle Inhalte durchsuchen)
6. Distortion Format und allgemein Formate in allen nötigen Scripten von Nuke und Maya Renderings anpassen
7. In Python einarbeiten → https://www.w3schools.com/python/default.asp

8. Alle Tools einmal testen, ob diese funktionieren oder Fehler auftreten und etwas an Skripten geändert werden muss und überprüfen, ob alle Skripte auch die Dateien im richtigen Ordner abspeichern.
9. RenderSequence Script und Playblast Script für Maya unter vfx_pipe\maya\python\fastRenderSequence bzw. vfx_pipe\maya\python\fancyPlayblast auf Wunsch des CG Supervisors anpassen, dass da alles im richtigen Format rauskommt
10. Comp Template anpassen an Wünsche des Lead Compers, sodass alle Artists sich nur das Template laden müssen und kleine Anpassungen in z.B. Farbe oder Licht machen und Rotos ziehen müssen. Hier ist es auch wichtig, dass man
    - Die richtige Grainplate reinlädt, also den Pfad der Grain Plate Read Nodes anpassen
    - Das richtige Format und Farbraum in den Read Nodes und allen Reformats einstellt
    - Den richtigen Farbraum in den Read Nodes einstellt
11. Vor Start der Postproduktion eine Einführung in die Pipeline für die Artists geben → wichtig da nochmal zu sagen, dass alle über RV hochladen sollen und nicht über anderen Weg, da sonst der Path to Frames/Files auf ShotGrid nicht stimmt
12. Die Datei shotliste.ini unter vfx_pipe\nuke\stupro\createNewScript anpassen, sodass dort die richtigem Shotbenennungen stehen, daraus holt sich das Contact Sheet Script die Namen und erstellt die Pfade zu den richtigen Dateien.
13. Erstes Contact Sheet erstellen und dafür einen Task auf ShotGrid erstellen (COSH_0000) sodass der Lead Comper nur noch diese Task aktualisieren (aktuelle jpegs oder exrs reinladen über Button), das ganze rendern und auf ShotGrid hochladen muss
14. Sich um die ToDos kümmern (https://gitlab.com/eldur_vfx/stupro_pipeline/-/issues) und diese uptodate halten, wenn neue Probleme auftreten.
15. Schauen was man in der Pipeline an Tools ungenutzt ist oder nicht funktioniert und das rausschmeißen (am besten erst in einen anderen Ordner verschieben um zu sehen, ob es wirklich unnötig war.
16. Optional: Remote einen Zugriff auf einen HdM PC bekommen, sodass man auch nach HdM Schließung Zugriff hat

Vorgehen Remote

In einem Remote Setup wird anstelle des Z Laufwerks an den Rechnern der Hochschule die Nextcloud verwendet. Dies bringt jedoch einige Probleme mit sich! Auch eine Mischung zwischen lokal und remote ist schwierig. Dies liegt hauptsächlich daran, dass in ShotGrid ein primary FilePath angegeben ist. Das ist der Pfad, wo alle Produktionsdateien liegen. Unterscheiden sich hier die Pfade zwischen Z und der Nextcloud, die beispielsweise auf D liegt, so kommt es zu Problemen. Um dies zu umgehen, müssten Personen, mit ‚falschen' Pfaden eine extra ShotGrid Konfiguration verwenden, welche andere Dateipfade nimmt. Arbeiten zwei Personen an einem Shot kann dies aber auch zu Problemen führen.

Unterschiede im Aufsetzen der Pipeline bestehen hauptsächlich in der ShotGrid Pipeline Konfiguration. Hier sollte eine dezentrale, distributed Konfiguration verwendet werden. D.h. dass die aktuelle Konfiguration beim Starten von ShotGrid runtergeladen wird und sie sich nicht auf einem gemeinsamen Laufwerk befindet. Dies kann beim Erstellen der Konfiguration über das Advanced Project Setup ausgewählt werden. Die Config ist dann als zip auf ShotGrid Web hochgeladen.

Die Dateistruktur der VFX StuPro baut auf dem Laufwerk Z: auf. Hier befinden sich alle Produktionsdaten sowie eben die Dateien für die Pipeline und ShotGrid. In dem ShotGrid Ordner liegt jeweils ein Ordner für jedes Projekt, in dem sich die dazugehörige ShotGrid Konfiguration befindet. Des Weiteren ist der `vfx-pipe` Ordner wichtig. Hier befinden sich die Tools für die jeweiligen DCCs drinnen, womit der Ordner das Kernstück der Pipeline darstellt.

Desweiteren ist die Pipeline auf Gitlab zu finden. Es bietet sich durchaus an, sie hier auch aktuell zu halten (siehe Kapitel Git): https://gitlab.com/eldur_vfx/stupro_pipeline

Die folgenden Kapitel sind angelehnt an die Ordnerstruktur des `vfx-pipe` Ordners. Außerhalb eines Ordners liegen hier auch noch die PIPELINE_CONSTANTS. In diesen werden die aktuellen Daten einer StuPro eingetragen und die einzelnen Tools lesen hieraus notwendige Informationen aus.

Nuke

Einstiegspunkt für das Scripting in Nuke sind die init.py und die menu.py. Hier können Einstellungen getroffen und Plugins hinzugefügt werden. In der Standard Konfiguration von Nuke werden diese Dateien in folgendem Ordner gesucht:

C:\Users\%USERNAME% \.nuke

Mit Hilfe der Umgebungsvariablen NUKE_PATH lässt sich dieser Ordner verändern.

In der hdm Pipeline gibt es zwei verschiedene Ordner für Nuke. Einmal handelt es sich um einen ganz allgemeinen Ordner mit Gizmos, die bei jedem Projekt vorhanden sein sollten und dann gibt es die Möglichkeit daneben einen weiteren Ordner anzulegen, in dem Projektspezifische Anpassungen vorgenommen werden können. Die Umgebungsvariable wird durch Peter oder im AdminSkript gesetzt. Als NUKE_PATH wird der ganze Nuke Ordner angegeben, wodurch sowohl die HDM Konfiguration als auch die spezifische geladen wird, ohne dass sie zu einer vermischt werden müssen. Damit dies funktioniert gibt es insgesamt drei init.py Files. Die init.py im nuke Ordner sorgt dafür, dass die beiden anderen aufgerufen werden.

Die Aufteilung des nuke/stupro Ordners ist wie folgt:

- Templates: beinhaltet das allgemeine Comp und Prep Setup sowie Distortionplates von den CP3 Optiken
- Toolset: .nk Dateien
- show_gizmos: .gizmo Dateien
- show_scripts: .py Dateien

Je nach Dateityp können in den jeweiligen Ordnern eigene Python Skripte oder Gizmos von Nukepedia ergänzt werden. Pythonskripte müssen dabei in der init.py oder menu.py integriert werden. Gizmos sollten auftauchen, sobald sie im richtigen Ordner liegen. Durch die Zeile nuke.pluginAddPath() weiß Nuke, wo nach den PlugIns gesucht werden soll.

Werden neue Tools für Nuke geschrieben, so geschieht das in der Regel in der Programmiersprache Python. Mit dem Script Editor von Nuke kann man diese gut debuggen. Wenn Nuke über Shotgun gestartet wird, so taucht die Kommandozeile von Nuke, die sonst im Hintergrund läuft nicht auf. In ihr stehen aber häufig wichtige Fehlermeldungen, wenn es Probleme mit der Pipeline gibt. Diese Fehlermeldungen, sind dann allerdings im Script Editor zu finden.

Im Folgenden wird auf einzelne, insbesondere selbst geschriebene Tools eingegangen.

init.py

Da die init.py auch geladen wird, wenn nur eine Nuke Kommandozeile geöffnet wird (dies ist beispielsweise bei Renderfarmen der Fall), werden in ihr wichtige Einstellungen getroffen. In der aktuellen init.py sind dies beispielsweise:

- Verwendung der OCIO
- Shot Setup Felder in den Project Settings hinzufügen
- Default-Verhalten von Nodes ändern
- Auflösungen der plates und der undistorted plates

Es ist auch möglich mit Callbacks zu arbeiten, sodass ein Skript z.B. immer ausgeführt wird, wenn die Nuke Datei gespeichert wird.

menu.py

Im Endeffekt verhält sie sich wie die init.py mit dem Unterschied, dass sie nur für GUI-Sessions aufgerufen wird. Hier wird die Funktionalitäten von Nuke um eigene erweitert, indem ein Menü mit verschiedenen Funktionen festgelegt wird. Die Nuke eigene Dokumentation ist hier gut als Einstieg geeignet: Custom Menus and Toolbars.

Autosave

Dieses Script ermöglicht einen rollenden Autosave, d.h. es werden autosave Dateien mit Ziffern von 0-9 gespeichert, ist die 9 erreicht, so wird wieder bei 0 begonnen und die alte Datei überschrieben.

kbCreateNewScript

In diesem Modul geht es sowohl darum einen neuen Shot zu starten, indem ein Template geladen wird, als auch darum immer die aktuellen Renderings in der Comp zu haben (geschriebenn von Katharina Böttcher). Wird ein Template geladen, so werden auch bereits automatisch die passenden Plates mitgeladen, damit der Artist diese nicht suchen muss. Ist der Shot dann einmal gestartet wird nur noch überprüft, ob es sich noch um die aktuelle Version des 3D Renderings handelt. Dazu werden Read Nodes, welchen einen bestimmten Namen (z.B. CG oder MP) haben, automatisch geupdatet.

Gegebenenfalls müssen hier Dateipfade angepasst werden, damit die Plates gefunden werden. Fehler können auftreten, wenn in einem Footage Ordner noch andere Dateien als die Bildsequenz liegen. Eine weitere Funktion ist das Setzen der Global Frame Range, diese Funktion hat zuletzt Fehler geworfen und sollte überarbeitet werden. Die erste und letzte Framenummer sollte aus den Dateinamen extrahiert werden. Eine Alternative wäre die Global Range in Shotgun einzutragen und dort auszulesen (TODO).

Ein weiteres Script in diesem Modul ist das kbCreateContactSheet. Hiermit können automatisch Contact Sheets mit beliebigen Shots erstellt werden. Die Shots werden in der Datei shotliste.ini konfiguriert.

CustomReadNode

Die Custom Read Node lädt ähnlich wie das CreateScript automatisch Footage ohne, dass der Artist den betreffenden Ordner suchen muss. Sie liest dafür den ScriptContext und baut selbstständig Pfade zusammen. Die Node wurde zuletzt nicht verwendet und benötigt gegebenenfalls Überarbeitung, da sie an sich durchaus sinnvoll ist.

CustomWriteNode

Diese Node fügt Funktionalitäten zu einer normalen Write Node hinzu. Lädt man also eine neue WriteNode, so hat diese automatisch die Pipeline-Funktionalitäten. Dazu gehört ein neuer Tab, in dem

man festlegen kann, nach welchem Pfadtemplate gerendert werden soll. Je nachdem wird dann direkt der richtige Pfad festgelegt und die Einstellungen für mov, jpg oder exr getroffen. Zudem ist in diesem Skript die Funktion beinhaltet, die den aktuellen Kontext anhand des Filenamens herausfindet. Die dabei erhaltenen Funktionen werden in den Project Settings unter Shot Setup gespeichert. Gegebenenfalls müssen hie Dateipfade angepasst werden. Falls benötigt können auch weitere Pfadtemplates hinzugefügt werden.

SaveNewCompVersion

Dieses Script überprüft beim Speichern, dass keine Ungereimtheiten entstehen. Es achtet also darauf, dass immer hoch versioniert wird, aber auch darauf, dass beim Hochversionen keine bereits vorhandene Version überschrieben wird.

Sonstiges

Weitere nützliche Tools, die in dieser Konfiguration bereits vorhanden sind, sind:

- Cryptomatte: Markieren von einzelnen Elementen eines 3D Renderings beispielweise aufgrund der Textur
- despill-madness: Entfernen von Spill bei Greenscreen Aufnahmen
- HdM Slice Tool
- mmColorTarget: Farbliches Anpassen einer Aufnahme auf die richtigen Farbwerte aufgrund eines aufgenommenen ColorChecker.
- Und vieles weitere…

Maya

Die Integration der Pipeline für Maya beginnt in Shotgun. Hier wird festgelegt, dass Maya mit einer bestimmten Zeile gestartet werden soll. Möchte man Maya mit eben dieser Konfiguration ohne Shotgun starten, so erfolgt dies über die Kommandozeile:

```
`call "c:\program files\autodesk\maya2020\bin\maya.exe" -script
"%PIPE_PIPELINEPATH%\lovebirds_pipeline\maya\scripts\launch_pipeline_tools.mel"`
```

Das aufgerufene Skript integriert dann das PipelineTools-Menü. Dieses Menü wird in der Datei `maya/python/setup/init_menu.py` definiert. Weitere Funktionen können hier ergänzt werden.

Neben einem neuen Menü können auch eigene Shelves erstellt werden. Dies geschieht in der Datei `maya/python/shelves/build_shelves.py`. Das Dev-Shelf ist hier so angelegt, dass nur bestimmte User es sehen können. Im Idealfall der Entwickler. Dazu muss der Name angegeben werden, mit dem man auf dem Rechner eingeloggt ist.

Die meisten externen Tools lassen sich auch einfach integrieren, indem man sie im `maya/python` Ordner speichert und die main-Funktion im Menü oder im Shelf aufrufen lässt. So ist es beispielweise mit den Tools AnimSchoolPicker2020, aTools und QuadFillHole passiert.

Neben Python kann man in Maya auch mit der eigenen Sprache MEL programmieren. Bisher wurde es so gehandhabt, dass notwendige MEL-Commands in Python umgewandelt wurden. Dies ist z.B. möglich mit dem Command `pymel.core.mel.eval()`.

DJV

DJV ist eine alternative, Open Source Review Software für Shots und ähnelt damit RV. Dieser Ordner ist wohl ein Überbleibsel aus alten StuPros, welcher nicht wirklich genutzt wird. Eine Dateisuche hat ergeben, dass DJV-View für die Skripte Background-Playblast und Render-Submit verwendet werden. Beide Tools wurden von Sam Kenworthy geschrieben und befinden sich im python Ordner. Daher im Python Kapitel mehr dazu.

ffmpeg

ffmpeg ermöglicht das Aufnehmen, Konvertieren und Filtern von Video- und Audioformaten. Ffmpeg wird genutzt vom background_playblast.

TODO: Beide Ordner (DJV und ffmpeg) sollten konkreter einem Tool zugeordnet werden oder aber in einen Lib Ordner verschoben werden.

ImageMagick

ImageMagick ist ein Programm um Vektor- und Rastergrafiken zu erstellen. Die Anwendung wird mit hoher Wahrscheinlichkeit nicht von Maya genutzt.

TODO: Es sollte getestet werden, ob der Ordner gelöscht werden kann.

lookdev_kit

Das Lookdev Kit lädt automatisch ein HDRI und eine Turntable Umgebung.

modules

Dieser Ordner beinhaltet externe Module für Maya, welche in MEL gescriptet sind.

- Exocortex: Alembic Export und Import
- OpenVDB: Verwandlung von Geometrie zu Volumes und andersrum.

python

Dieser Ordner beinhaltet Tools für Maya, welche in Python gescriptet sind.

- **animBot**: externes Tool zur Unterstützung bei der Animation
- **AnimSchoolPicker2020**: externes Tool zur Interaktion mit 3D Charakteren
- **aTools**: externes Tool zur Unterstützung bei der Animation
- **autosave_reminder**: Das Tool fragt in regelmäßigen Abständen nach, ob die Szene gespeichert werden soll. So kann verhindert werden, dass bei einem Maya Absturz Daten verloren gehen.
- **background_playblast**: Tool zum Rendern von Playblast geschrieben von Samuel Kenworthy an der hdm. TODO: Das Tool wirft aktuell einen Fehler, weswegen der fancyPlayblast stattdessen verwendet wird.
- **color_assigner**: Tool zum Einfärben von Objekten, ebenfalls programmiert von Samuel Kenworthy.
- **EzMel2Py**:
- **fancyPlayblast**: Tool zum Rendern von Playblasts
- **fastRenderSequence**: Das Tool setzt automatisch die gewünschten Render Settings. Die Settings sind hardgecodet in den Quellcode. Auch der Pfad, wohin die Bilder gerendert werden, wird hier in einer Funktion erstellt. Dieser muss gegebenfalls an die Ordnerstruktur des aktuellen Projekts angepasst werden. TODO: Settings und Pfaderstellung besser konfigurierbar machen. Beispielweise in den PIPELINE_CONSTANTS und ein extra Modul zur Pfadberechnung, welches von verschiedenen PlugIns genutzt werden kann.
- **fbx_export**: Ein schnellerer Weg um entweder alles oder selektierte Objekte als fbx zu exportieren. Auch hier gibt es eine Funktion, die den Pfad des fbx ausrechnet. TODO: Pfadberechnung auslagern, um sie besser konfigurierbar zu machen.
- **mutils**: Ein Teil des externen PlugIns Studio Library. Das Tool ermöglicht das Verwalten von Posen und Animationen in Maya.
- **nodes**: Hier werden Plugins geladen.
- **pyblish_main**: Pyblish ist ein Tool, welches es ermöglicht die Struktur einer Maya Szene vor einem Publish zu validieren. Dazu gehören beispielsweise, die Kontrolle, ob alle Rig Teile in

einem Ordner Ordner Rig liegen. Oder aber auch, um zu Überprüfen, ob alle Render Settings richtig eingestellt sind. Die Funktionalität wurde zuletzt für Zwischen Mehl und Milch verwendet. Die Verwendung hängt davon ab, wie wichtig dem CG Supervisor eine Einheitlichkeit bei den 3D Szenen ist. Die Dokumentation ist hier zu finden: https://api.pyblish.com/

- **QuadFillHole:** Externes Tool zum Füllen von Löchern in der Geometrie. Eine genaue Beschreibung steht in der Read-Me.txtx.
- **render_checker:** Wie der Name sagt, ein Tool, welches Renderings überprüft (geschrieben von Sam Kenworthy) → wurde seit Zwischen Mehl und Milch nicht mehr genutzt
- **render_Submitter:** Das Plugin (geschrieben von Sam Kenworthy) schickt einen Render Job mit verschiedenen Layern zu RenderPal → wurde seit Zwischen Mehl und Milch nicht mehr genutzt
- **rolling_plugin:**
- **setup:** Hier werden die Menus in Maya definiert
- **shelves:** Hier weden die Shelves in Maya definiert
- **studiolibrary:** Ein Teil des externen PlugIns Studio Library. Das Tool ermöglicht das Verwalten von Posen und Animationen in Maya.
- **studiolibrarymaya:** Ein Teil des externen PlugIns Studio Library. Das Tool ermöglicht das Verwalten von Posen und Animationen in Maya.
- **studioqt:** Ein Teil des externen PlugIns Studio Library. Das Tool ermöglicht das Verwalten von Posen und Animationen in Maya.
- **studiovendor:** Ein Teil des externen PlugIns Studio Library. Das Tool ermöglicht das Verwalten von Posen und Animationen in Maya.
- **utils:** Kleinere nützliche Skripte um beispielweise nicht genutzte Texturen zu löschen oder das LookDev Kit zu laden.
- **config.json**

scripts

Dieser Ordner beinhaltet ein einziges Skript. Es sollte beim Starten von Maya ausgeführt werden, um die custom Tools und Shelves zu laden.

Pyblish

Quickstart: https://gist.github.com/mottosso/93399862c94f0ab4314f https://learn.pyblish.com/03-files

Für die Installation muss Python, PySide2 und PyQt5 auf dem Rechner installiert sein. Dies geschieht einfach über pip install in der Commandline.

```
pip install PyQt5

pip install PySide2
```

Falls pip nicht funktioniert, muss es separat installiert werden https://phoenixnap.com/kb/install-pip-windows. Um pyblish zu installieren, muss folgendes in die Kommandozeile von Windows eingegeben werden:

```
mkdir pyblish                          # Verzeichnis am richtigen Ort erstellen
cd pyblish
pip install pyblish --target .
set PYTHONPATH=%cd%                    # Umgebungsvariable setzen %cd% gibt den aktuellen
                                       # Ordner zurück, dies muss der pyblish Ordner sein
python -m pyblish_qml --demo           # Pyblish-Qml Demo öffnen zum Test
set PYTHONPATH=%cd%;%cd%\pyblish_maya\pythonpath
```

```
set PYBLISH_GUI=pyblish_qml
set PYBLISH_QML_PYTHON_EXECUTABLE=c:\python37\python.exe
call "c:\program files\autodesk\maya2020\bin\maya.exe"
```

Hier findet man die Dokumentation zu dem Ablauf: https://forums.pyblish.com/t/trying-to-set-up-pyblish-and-running-into-common-problems-all-typical-users-hit/500/2.

Um PlugIns zu registrieren, muss wieder eine Umgebungsvariable gesetzt werden, die angibt, wo die Skripte gespeichert werden:

```
set PYBLISHPLUGINPATH=C:\Users\Katha\PycharmProjects\lb_pipeline\pyblish\plugins
```

Im File Menü von Maya sollte nun ein Reiter "Publish" sein und sich dann ein pyblish-Fenster öffnen.

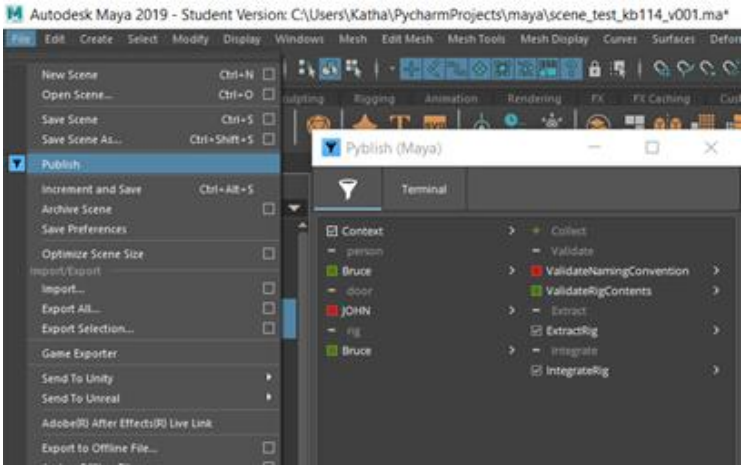

Abbildung 4: Pyblish öffnen in Maya. Quelle: Eigene Abbildung.

Es können beliebig PlugIns für Pyblish ergänzt werden. Im der folgenden Abbildung sieht man, wie so ein PlugIn beispielsweise aufgebaut sein kann:

```
class ValidateRigContents(pyblish.api.InstancePlugin):
    """Ensure rig has the appropriate object sets"""

    order = pyblish.api.ValidatorOrder    #order = 0 usw legt Reihenfolge fest
    families = ["rig"]

    def process(self, instance):
        assert "controls_SEL" in instance, "%s is missing a controls set" %
                                                    instance
        assert "pointcache_SEL" in instance, "%s is missing a pointcache set"
                                                    % instance
```

Abbildung 5: Beispielaufbau eines Pyblish PlugIns. Quelle: Eigene Abbildung.

ShotGrid

ShotGrid ist ein datenbankbasiertes Tool zur Verwaltung und Organisation von Shots und Aufgaben in einem Projekt. Zudem können Reviews auf Shots vergeben werden. Man unterscheidet zwischen der Web- und der Desktopversion von Shotgun. Die Webversion zeigt aktuelle Stände der Shots, ermöglicht es Reviews zu geben und Zeiten für Tasks zu verwalten. In der Desktopversion ist es möglich Programme zu öffnen, um dann die Task auszuwählen an der man am Arbeiten ist und so auf die Skripte von Nuke oder Maya zuzugreifen.

Aus Developer Sicht sind folgende Dinge wichtig:

- Softwareverwaltung
- Pipeline Configuration mit möglichen zusätzlichen Plugins

ShotGrid Configuration

Am Anfang einer Studioproduktion wird zunächst ein neues Projekt angelegt. Nach dem Anlegen des Projekts in Shotgrid Web sollte Shotgun Desktop geöffnet werden. Bisher gibt es noch keine Pipeline Konfiguration für dieses Projekt, wird nun also ein DCC über Shotgrid Desktop geöffnet, so ist die Shotgrid Integration unvollständig und man wird kein File für eine Task öffnen können. Die Pipeline Konfiguration wird am besten über das Advanced Project Setup erstellt.

Dazu einfach über Shotgun Desktop in das Projekt gehen und auf das Profilbild klicken, dann "Advanced Project Setup" wählen. Hier kann entweder eine vorhandene Konfiguration reingeladen werden, die von einem alten Projekt gewählt werden oder eine neue erstellt werden. In der Regel bietet es sich an, die Konfiguration des letzten Projekts zu übernehmen. Als Shotgun Configuration wird bei Remote Work das Distributed Setup verwendet. Im VFX Büro selbst ist dies nicht so sinnvoll, da sollte die Configuration auf einem Netzlaufwerk liegen (Z:\pipeline\shotgun).

Es ist sinnvoll das Advanced Project Setup zu durchlaufen und nicht einfach eine Configuration für das Projekt zu hinterlegen. In dem Setup werden auch Shotgun-spezifische Werte gesetzt, wie z.B. die Shotgun-ID und der sogenannte Tank (_PROD). Möchte man eine neue anlegen, muss man in Shotgun Web die alte löschen und ebenfalls bei den Eigenschaften des Projektes den Eintrag bei Tank löschen (Löschbar unter Overview am rechten Rand bei den Informationen). Was aber beispielweise aus dieser Config übernommen werden könnte, sind Templates (Templates.yml), also wie die Ordnerstruktur aufgebaut ist und wie die Namenskonvention ist.

Es gibt ein paar wichtige Dateien in der Shotgun Configuration, wo Sachen geändert werden können oder sollten:

1. core/Templates.yml: Hier werden die Templates festgelegt. D.h. hier kann man festlegen, wie die Namenskonventionen sind und wo was gespeichert werden soll. Dabei muss aufgepasst werden, dass es mit dem Ordner Schema übereinstimmt. Dieser ist eine Repräsentation wie die Ordnerstruktur im Projekt aussehen wird.
2. core/roots.yml: Festlegung der roots Dateipfade (z.B. primary auf Z:)
3. env/includes/...: In den hier liegenden Dateien wird festgelegt, wo die Skripte von Tools liegen. Die meisten Tools werden ShotGrid intern aus dem AppStore geladen, schreibt man eigene Toolkit PlugIns, so kann hier auch ein Dateipfad oder eine Git URL angegeben werden.

```
apps.tk-multi-publish2.location:
  type: app_store
  name: tk-multi-publish2
  version: v2.6.1
```

```
apps.custom-app.location:
  type: dev
  name: tk-multi-publish2
  path: Z:/pipeline/custom-app
```

4. env/includes/settings/...: Hier werden Einstellungen für die verschiedenen Plugins getroffen. In der Datei tk-maya.yml wird beispielweise festgelegt, dass im Context Shot Step im ShotGrid Menu die Option publish auftauchen wird. In anderen Kontexten existiert dieser Menupunkt hingegen nicht.

Software Einstellungen

Als Admin in Shotgun hat man ein zusätzliches Menü, wenn man auf der Webseite auf sein Icon klickt. Hier drunter ist der Reiter Software zu finden. Damit kann konfiguriert werden, wie Shotgun welche Programme öffnet. Es ist also möglich die Version zu bestimmen und auch ob beim Öffnen ein Skript ausgeführt werden soll.

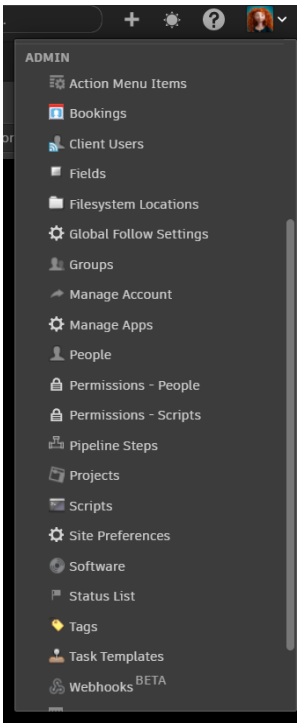

Abbildung 6: Software Verwaltung aufrufen. Quelle: Eigene Abbildung.

Bei Maya werden in unserer Configuration dabei Windows Args angegeben mit denen Maya geöffnet werden soll. Dies triggert, dass Maya mit den PipelineTools geöffnet wird. Öffnet man Maya also ohne Shotgrid, so werden diese Tools nicht auftauchen! Die Windows Args lauten wie folgt:

```
-script
"%PIPE_PIPELINEPATH%\lovebirds_pipeline\maya\scripts\launch_pipeline_tools.mel"
```

Startet man Maya mit einer Kommandozeile und diesem Argument hinter der maya.exe, so sollten die Tools auch auftauchen.

Shotgun API

Mithilfe der Shotgun API kann in einem Python Script auf Informationen innerhalb von Shotgun zugegriffen werden. Man kann zum Beispiel Shots finden und nach bestimmten Sachen filtern, oder auch neue Versionen anlegen. man kann auch auslesen, welche Frame Range eines Shots im Schnitt verwendet wird um diese Information als Global Range in Nuke zu verwenden. Um die API verwenden zu können, muss man in Python das Package shotgun_api3 importieren. Wie dieses installiert wird, findet man hier. Beispiele zur Verwendung der API findet man hier.

Im Groben muss zunächst eine ShotGrid Instanz erstellt werden. Hierbei muss man sich entweder mit einem Script Key oder dem UserName verifizieren. Im Anschluss ist es möglich mit der Funktion find oder find_one() Shots oder Assets zu suchen. Auf den Shots können dann alle Informationen abgefragt werden. Es können so auch Shots gepublished oder neue Shots erstellt werden. Der Fantasie sind hier wenige Grenzen gesetzt.

Shotgun Toolkit

Shotgun Toolkit ist die Umgebung, in der programmiert wird, wenn man beispielweise Apps für ShotGrid Desktop (in Python) schreibt. Aber auch die bereits vorhanden Funktionen, wie file open oder publish fallen unter das toolkit. Der Code hinter diesen Funktionen kann auf GitHub (https://github.com/shotgunsoftware) eingesehen werden. Eine weitere Möglichkeit von Toolkit neben Apps in ShotGrid Desktop sind Action Menu Items. Dieser können in einem Rechtsklick Menu in Shotgrid Web beispielsweise auf einem Task aufgerufen werden und sind kontextsensitiv.

Eine Einführung in die Programmierung mit Toolkit findet man hier:
https://developer.shotgridsoftware.com/tk-core/overview.html#what-is-the-toolkit-platform

Bisher gibt es keine StuPro PlugIns, die in Toolkit laufen. An sich ist hier der Aufbau der Konfiguration wichtiger.

Weiterführende Links

- Dokumentation mit Beispielen
- Information zum Config Setup

OCIO

OCIO steht für Open Color IO und ist ein Tool fürs Color Management. Color Management ist wichtig, damit die Farben, die im 3D gerendert werden, im Compositing auch genauso ankommen und genauso rausgehen. Beliebte Konfigurationen sind ACES oder Filmlight. Zu Beginn des Projekts sollte sich hier auf eine Konfiguration geeinigt werden, die dann in allen Programmen genutzt wird.

Sonstiges

CopyToGrading

REMOTE PIPELINE ONLY!

Das Skript kopiert die aktuellen Comp Versionen in den copy-to-grading Ordner. Dieser ist ein externer File Storage und führt auf das VFX-NAS, sodass vom Baselight System aus darauf zugegriffen werden kann. Das Skript überprüft dabei, dass auch immer nur der aktuellste Ordner auf dem VFX-NAS liegt um den Speicherplatz klein zu halten. Aufgrund das Baselight Versionen erkennen kann, wird in dieser Fassung nicht Kürzel und Versionsnummer weggelassen, wie es in früheren Versionen der Fall war. Aktuell darf sich, damit Baselight das File erkennt, aber nur die Versionsnummer und nicht das Kürzel ändern! Ansonsten muss der Clip manuell gerelinked werden. Das Skript wird ausgeführt indem man eine Kommandozeile öffnet und in den Ordner wechselt, wo das Skript liegt. Dann kann es mit folgendem Befehl ausgeführt werden: `python copyToGrading.py`

PetersAdminScript

REMOTE PIPELINE ONLY!

Dieses Skript setzt Umgebungsvariablen ausgehend, von dem Ort, wo das Skript liegt. Alle Pfade werden also relativ angegeben, zum Ort des Skriptes. Dies hat den Vorteil, dass es egal ist, auf welchem Laufwerk die Nextcloud gespeichert ist. Es wird regelmäßig auf allen Hochschulrechnern ausgeführt, so dass Änderungen auch recht schnell beim Artist übernommen werden.

Git Einführung

Git ist ein Programm zur Versionskontrolle. Hier wird aktualisierter Code hochgeladen und es kann aber auch immer auf alte Stände zurückgegriffen werden. Wird beispielsweise ein Tool gelöscht, welches später doch benötigt wird, so kann hier noch darauf zu gegriffen werden. Scripting Projkete unter Version Control zu halten ist immer sinnvoll. Häufig ändert man Codezeilen, um einen Fehler beheben und stellt dann fest, dass der Fehler doch einer anderen Stelle entsteht. Sich dann zu erinnern, welche Zeilen Code man wo geändert hat und was da vorher stand ist schwierig. Version Control emöglicht es jetzt zu sehen, welche Dateien geändert wurden und einen einfachen Rollback zu machen, bei den Dateien, wo die Änderungen unnötig war. Desweiteren können Branches erzeugt werden, um an spezifischen Sachen zu arbeiten, ohne den master Branch zu verändern. Änderungen, die dann final sind, können einfach in den master branch gemerged werden.

Im folgenden zwei Varianten um git zu verwenden:

- git kann aus der Kommandozeile aufgerufen werden. Dazu muss git zunächst installiert werden. Mit Rechtsklick in dem Ordner, welcher das Git Repo ist, kann dann Git Bash here oder Git Gui ausgewählt werden. Git Gui ist das User Interface zur Kommandozeile, auch hier

können alle Funktionen von git angewendet werden. Git Bash öffnet eine Kommandozeile. Wichtige Kommandos, die man dort eingeben kann sind:

- `git add .` → added alle Dateien zur Version Control, statt . kann eine spezifische Datei angegeben werden
- `git commit -m ‚commit message'` → commit mit aussagekräftiger Nachricht
- `git checkout -b branchname` → wechseln auf einen anderen branch branchname, -b erzeugt einen neuen branch, falls es den Namen nicht gibt.
- `git pull` → runterladen des aktuellen Stands auf gitlab
- `git push origin branchname` → Hochladen des aktuellen lokalen Rep Stand auf den branch branchname

- git kann aber auch in PyCharm verwendet werden. Es kann sein, dass man in den Einstellungen erst festlegen muss, welche Ordner unter Versionskontrolle liegen. Dazu gibt es in den Einstellungen eine Seite Version Control, wo Pfade und die Art der Version Control, in dem Fall git angegeben werden können. Klingt man dann mit der rechten Maustaste auf den Ordner in PyCharm, so gibt es den Unterpunkt git. Hier können Dateien geadded werden, commits abgeschickt werden und der aktuelle Stand gepushed oder gepulled werden.

➔ Auf dem Master Branch befindet sich ein allgemeines Pipeline Setup. Projektspezifische Änderungen sollten auf eigenen Branches erfolgen!