

2016

ISSN 1433-2620 > B 43362 >> 20. Jahrgang >>> www.digitalproduction.com

Deutschland € 15,20

Published by ATEC

Österreich € 17,-

Schweiz sfr 23,-

2

DIGITAL PRODUCTION

DIGITAL PRODUCTION

MAGAZIN FÜR DIGITALE MEDIENPRODUKTION

MÄRZ | APRIL 02:2016



Fokus: Pipelines

Back-ups & Asset-Management-Grundlagen, Tools & Tricks

Zoomania

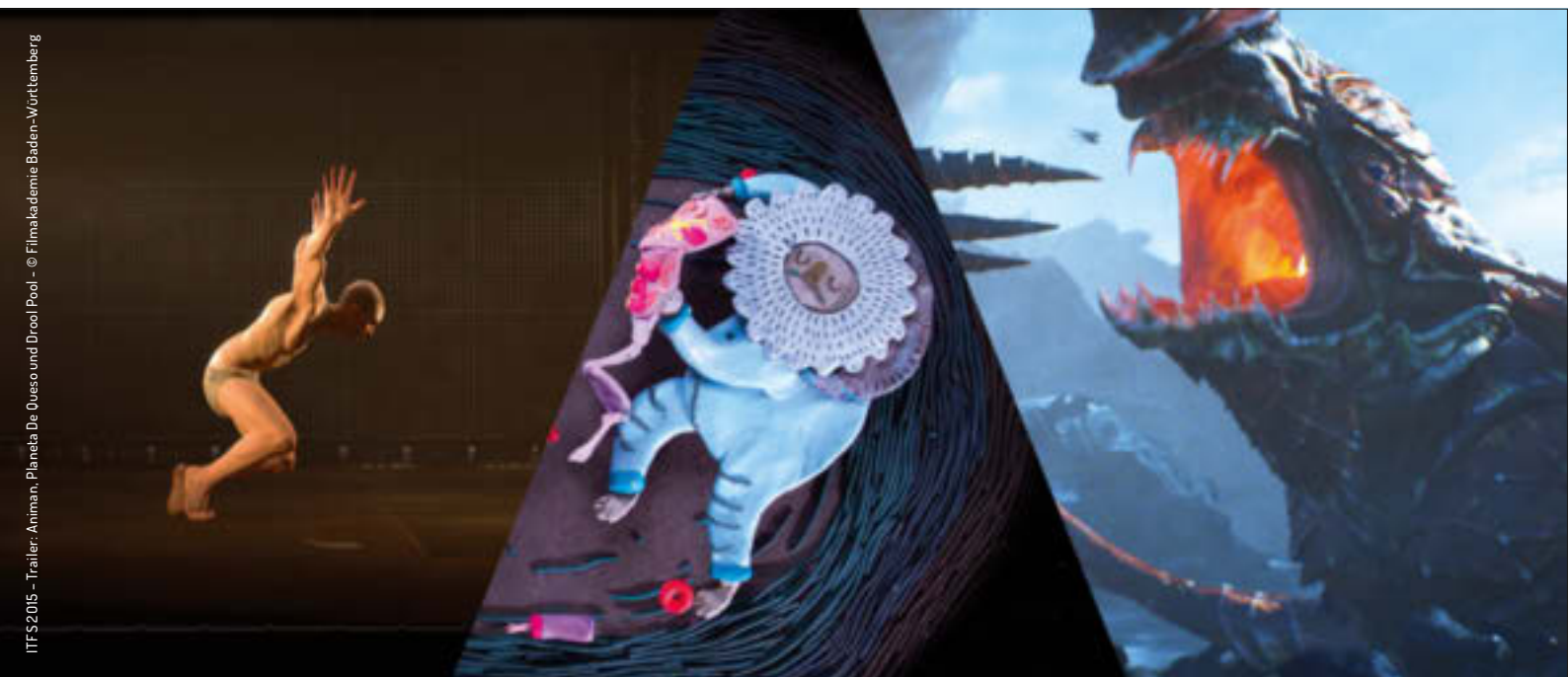
Kinorunde mit Star Wars, Disney und The Martian

Praxis satt!

4K-Monitore im Test, Quixel, Houdini Engine, Sony A7R II



4 194336 215200 02



Pipelines für Animations- und VFX-Produktionen

Im Zuge des Fortschritts wird nicht nur die Hardware, sondern auch die Software, die wir nutzen, performanter. Damit entfernen wir uns weiter von den Basisprozessen einer Produktion und sind gezwungen, kleinere Chunks in größeren Verkettungen zu übernehmen, um noch höhere Anforderungen in noch kürzerer Zeit zu bewältigen. Dieser Bewegung stellen sich Pipelines entgegen, die versuchen, Ordnung in das Chaos zu bringen – nicht ohne schnell selbst das Chaos zu verursachen.

von Alexander Richter

Das Wort Pipeline setzt sich aus den Worten „Pipe“ und „Line“ zusammen und beschreibt einen geradlinigen Vorgang, bei dem auf der einen Seite etwas reingegeben wird, das verändert auf der anderen Seite wieder herauskommt. Pipelines kann es für alles geben – von der Render-Pipeline einer 3D-Software, die eine Projektion dreidimensionaler Objekte per Raytracing zweidimensional darstellt, bis hin zu einer Produktions-Pipeline, die regelt, wie der Workflow und die Datenausschüttung in den einzelnen Phasen zu sein hat.

Die Erstellung einer solchen lässt sich in die Phasen der Verzeichnisstruktur und Datenverwaltung, Namenskonvention, Software- und Skript-Pipeline aufspalten.

Verzeichnisstruktur

Auf der untersten Ebene der Strukturierung ist die Datenverwaltung und Verzeichnishierarchie. Die Verzeichnisebene entscheidet fundamental über die Arbeitsweise und Handhabung des Projekts und sollte im Einzelnen getestet werden. Dabei haben sich folgende Kategorien herauskristallisiert:

1. Data

- Pipeline (Plug-ins, Skripte, Icons)
- User Space (privater Bereich für Tests, um nichts auf dem Desktop zu speichern)
- Footage (Plates, HDRIs, Texturbibliothek)

2. Preproduction

- Animatic
- Reference
- Storyboard
- Research
- Concept & Mood
- Script/Story

3. Production

- Assets (Chair/Artist, Prop/Chair, Set/Street)
- Shots (010_shotName, 020_newShotName ...)
- Renderings

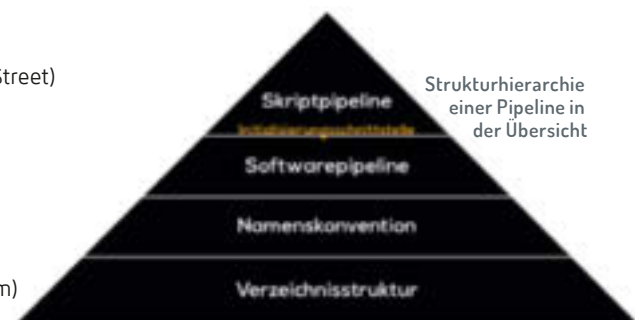
4. Postproduction

- Compositing
- Editing (Sound, Music, Cut, Film)
- Grading

5. Organisation Space

- Finance
- Presentation
- Planning
- Making-of
- Promo (Presse-, Projektmappe)
- Dailies/Weeklies

Die Gewichtung und Notwendigkeit der einzelnen Bereiche ist abhängig von dem Projekt, von seiner Komplexität, Größe und Teamstärke. Kleinere Projekte werden sich schwer tun, die ganze Bandbreite und Tiefe dieser Strukturierung zu nutzen. Größere Projekte dagegen gehen schnell in der Komplexität ihrer Aufteilung unter und sind kaum durchschaubar.



Wichtig ist es, zu Beginn zu wissen, ob sich ein Mensch oder eine Software durch die Struktur bewegen soll. Ruft ein Skript die Arbeitsdateien ab, speichert sie und setzt Pfade, dann ist Konsistenz das Fundament einer Struktur. Muss sich der Mensch mit der Struktur beschäftigen, sind Ausnahmen, die Lesbarkeit und Einfachheit verbessern, angebracht.

Als Beispiel: Es gibt ein Verzeichnis für Charaktere, Props und Sets, aber nur ein Straßenset. Die Softwarelösung wäre, das Straßenset unter dem Verzeichnis „Set“ abzuspeichern, damit der Pfad konsequent bleibt (Set/Street). Die Nutzerlösung dagegen würde das Straßenset direkt, ohne ein Zwischenverzeichnis, abspeichern (Street).

Wichtig ist, sich auch Gedanken über Back-ups zu machen, um zumindest die wichtigsten Daten abzusichern. Eine Versionierung bietet die Möglichkeit, inhaltliche Fehler zu korrigieren, aber sie bewahrt nicht vor einer physikalischen Zerstörung. Ein RAID5-System würde sich anbieten, da es nicht nur die Zugriffsgeschwindigkeit auf die Dateien erhöht, sondern auch eine Wiederherstellung erlaubt.

Zusätzlich bieten sich Repository-Systeme wie Git und Tortoise oder bei richtiger Nutzung auch Dropbox an. Im Zweifel einfach auf die gute alte Backup-Festplatte zurückgreifen, die nicht selten mit einer internen Software daherkommt, die regelmäßige und automatische Updates ausführt.

Namenskonventionen

Neben der Struktur ist die Benennung entscheidend zur Orientierung und zum Informationsaustausch. Dabei wird unterschieden zwischen Arbeitsdateien (WORK) und Referenzdatei (PUBLISH). Die Arbeitsdatei muss versionierbar sein und sollte alle wichtigen Informationen, priorisiert in der Reihenfolge, im Dateinamen enthalten:

- ▷ Was ist der Inhalt? (Name)
- ▷ Welcher Aufgabenbereich wird abgedeckt? (Task)
- ▷ Um welche Iteration handelt es sich? (v001)
- ▷ Wer hat die Datei bearbeitet? (Author)
- ▷ Gibt es Besonderheiten? (Comment)

name_TASK_version_author_comment.format
 cat_RIG_v001_ar_brokenBones.mb

Die Referenzdatei (PUBLISH) sollte nur die nötigsten Informationen enthalten, da sie nicht verändert, sondern regelmäßig überschrieben und von anderen Stellen referenziert wird.

cat_RIG.mb

Ebenfalls sehr beliebt sind Namespaces statt oder ergänzend zum Namen, da sie kürzer sind und schneller wiedererkannt werden. Während sich diese gerade zur Gruppierung einzelner Bestandteile eignen, ist von einer Umwandlung oder zusätzlichen Nutzung ge-

rade im Dateinamen abzuraten. Das Resultat kann das Verständnis verkomplizieren, zu Dopplungen führen und den Mehrwert überschatten, der durch Pfad oder Dateiname gleichsam erhellt werden könnte.

Es bietet sich natürlich an, diesen Prozess durch Skripte zu steuern und dem Nutzer die Merk-Akrobatik abzunehmen. Speicher- und Lade-Skripte können unnötige Einarbeitungszeit reduzieren. Das Netz bietet diesbezüglich einige Vorlagen, die zum Beispiel die Versions-Inkrementierung beim Speichern übernehmen (v001 > v002). Maya und Nuke besitzen seit einigen Versionen den „Increment & Save“-Befehl. Woran diese Konzepte jedoch oft kränkeln, ist die Flexibilität, sich an andere Pipeline-Formen anzupassen. Eigene Skripte bieten sich hier an und sind schnell zu implementieren – immer vorausgesetzt, die Strukturierung leidet nicht unter Inkonzsequenz.

Software-Pipeline

Die Software-Pipeline beschreibt, welche Software an welcher Stelle des Projekts genutzt wird und was die Schnittstellen zwischen ihnen sind. Wichtig ist, dabei zusätzlich die Formate zu definieren: Von welcher Stelle und in welchem Format wird die Datei in den nächsten Task überführt oder aus welchen Bildformaten setzen sich die Texturen zusammen. Die Versionen der

WORK (ASSET/SHOT)
 mother MODEL v001 ar mirrored job
 RIG/SHOT 123 10200 0012 000001 10000

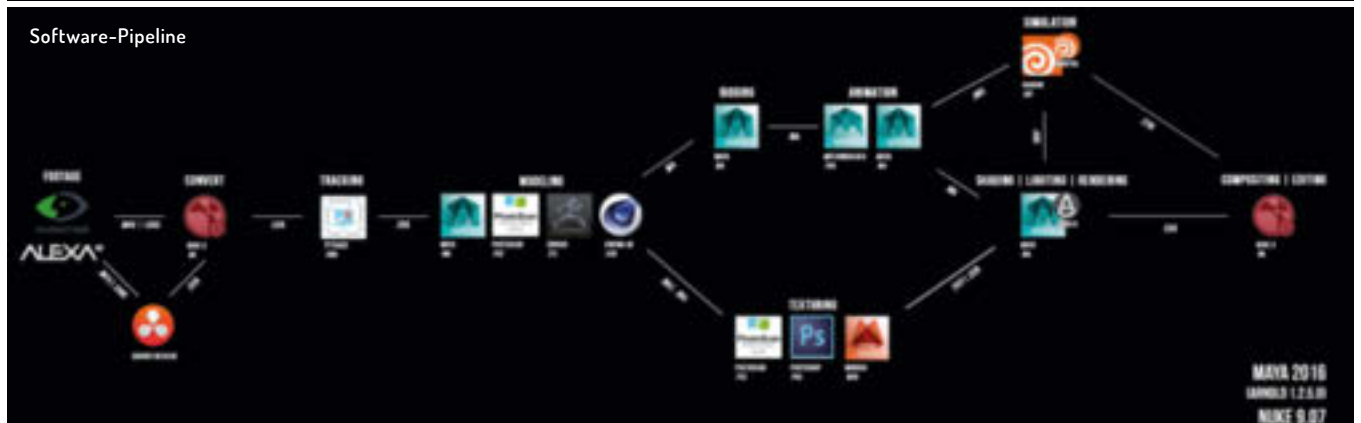
PUBLISH (ASSET/SHOT)
 mother SHOT .mb
 RIG/SHOT 123 10000

RENDERING
 010 AlexRigPlane 0001 .ar
 SHOT 001 10000 10000
 010_AlexRigPlane.0001.ar

TASK	TYPE
MODEL	MODEL
RIG	MODEL
TEXT	TEXTURE
RIG	MODEL
AR	ARBITRARY
LIGHT	LIGHTING
RIGS	MODEL
SH	SIMULATOR
FX	EFFECT
PREV1	SLAYCOMP
COMP	COMPOSITING

- english
- no spaces instead "_"
- version: v and 3 digits
- names are initials
- [mother_RIG]
- [v001]
- [Alexander Richter -> ar]
- lower CamelCase [brokenStone]
- Task are written in UPPERCASE [LIGHT]
- comments are optional

Namenskonventionen



Batch-Datei Maya.bat

```

1 @echo off
2 rem MAYA
3
4 rem --- Path ---
5 set "SOFTWARE_PATH=P:/pipeline/maya"
6 set "SHELF_PATH=%SOFTWARE_PATH%/shelf"
7 set "MAYA_VERSION=2016"
8
9 rem --- Python ---
10 set "PYTHONPATH=%SOFTWARE_PATH%;%PYTHONPATH%"
11
12 rem --- Shelf ---
13 set "MAYA_SHELF_PATH=%SHELF_PATH%;%MAYA_SHELF_PATH%"
14
15 rem --- Disable Report ---
16 set "MAYA_DISABLE_CIP=1"
17 set "MAYA_DISABLE_CER=1"
18
19 rem --- SplashScreen ---
20 rem File: MayaSD\startuelImage.png
21 set "XBIPLANGPATH=%SOFTWARE_PATH%/img/logo;%XBIPLANGPATH%"
22
23 rem --- Call Maya ---
24 set "MAYA_DIR=C:/Program Files/Autodesk/Maya/MAYA_VERSION/"
25 set "PATH=%MAYA_DIR%/bin;%PATH%"
26 start maya
27
28 exit

```

Batch-Datei für Maya, die zusätzliche Pfade für Python und Shelf implementiert und im Anschluss Maya startet. „Disable Report“ beschleunigt das Schließen der Software, was seit Maya 2016 obsolet geworden zu sein scheint.

Batch-Datei Nuke.bat

```

1 @echo off
2 rem nuke
3
4 rem --- Path ---
5 set "SOFTWARE_PATH=P:/pipeline/nuke"
6
7 set "NUKE_VERSION=Nuke9.0v7"
8
9 rem --- Python ---
10 set "NUKE_PATH=%SOFTWARE_PATH%;%NUKE_PATH%"
11
12 rem --- Init & Menu ---
13 set "NUKE_INIT_PATH=%SOFTWARE_PATH%;%NUKE_PATH%"
14 set "NUKE_MENU_PATH=%SOFTWARE_PATH%;%NUKE_PATH%"
15
16 rem --- Call Nuke ---
17 set "NUKE_DIR=C:/Program Files/NUKE_VERSION/"
18 set "PATH=%NUKE_DIR%;%PATH%"
19 start Nuke9.0.exe --nukex %1

```

Batch-Datei für Nuke, die zusätzliche Pfade für Init und Menu implementiert und Nuke startet.

Softwares (Maya 2016) und Plug-ins (Arnold 1.2.5.0) sollten in einer Tabelle oder in der Grafik festgehalten werden, um nicht nur transparent zu bleiben, sondern auch eine Konsistenz zu gewährleisten. Klarheit schafft nicht nur gleiche Bedingungen, sondern vereinfacht auch das kreative und technische Arbeiten, wenn die Beteiligten dieselben Werkzeuge nutzen.

Initialisierungsschnittstelle zur Skript-Pipeline

Nachdem eine sinnvolle Verzeichnisstruktur für das Projekt gefunden wurde und sich

fung der Software auszuführen. Dieser Weg erlaubt es auch, Plug-ins für alle Projektbeteiligten synchron zu halten und Konsistenz in Software und Versionierung zu gewährleisten.

Init & Menu

Grundsätzlich besitzt eine industrielle 3D- und Compositingsoftware Initialisierungsschnittstellen, die beim Start aufgerufen werden. Der Pipeliner kann sich diese zu nutzen machen, um Einstellungen zu setzen, Menüs zu erstellen oder sein Skriptsystem zu aktivieren.

herauskristallisiert hat, wie, wohin, als was und wann jeder Prozess gespeichert wird, benötigt es im Folgenden eine Pipeline-Schnittstelle.

Dies erlaubt ein Entwickeln und Erweitern im Hintergrund, ohne von dem Nutzer ein bestimmtes Verhalten zu verlangen. Nun liegt es am Projekt zu sagen, ob nur kleine Modifikationen wie Standardeinstellungen (Auflösung, Frame-rate etc.) definiert werden oder komplexere Programme nötig sind, die das Speichern, Exportieren und Arbeiten vereinfachen.

Dabei ist es zunächst wichtig, die Skript- und Plug-in-Pfade auf die Projektpipeline zu richten. Dies kann unter Windows durch das Erstellen einer Batchdatei (*.bat) verwirklicht werden, welche die Pfade der jeweiligen Software per Unix-Shell-Befehl anpasst.

Der Nutzer selbst bekommt eine Verknüpfung mit Icon und ist lediglich aufgefordert, diese anstatt der Standard-Startverknüpfung

Python und Py Side

Vor der Entwicklung eines Systems stellt sich zuerst die Frage nach der Programmiersprache. Dabei gibt es eine klare Antwort: Python. Python hat sich in den letzten Jahren als vielseitigste und am meisten genutzte Sprache im Animationsraum etabliert. Die Vorteile liegen auf der Hand: Sie ist einfach zu erlernen, eine interpretierte Programmiersprache (nicht kompiliert) und in den gängigen Softwarepaketen integriert (Maya, Nuke, Houdini, 3ds Max, Blender, C4D). Ergänzend dazu bietet sich die grafische Oberfläche von PySide an, die fest in die Struktur von Maya, Nuke und Houdini eingegliedert ist. Diese vereinfacht nicht nur das Erstellen einer software-unabhängigen GUI für die Skripte, sondern macht es dem Artist einfacher sie zu nutzen – durch einheitlichen Stil und Aufbau.

Skript-Pipeline

Nachdem die Schnittstelle zwischen Pipeline und Software hergestellt ist, muss diese nun mit Leben gefüllt werden. Der Grundgedanke einer Skript-Pipeline ist, die einzelnen Arbeitsprozesse einer Produktion „einfach und möglich“ zu machen.

„Einfach“ befasst sich zumeist mit der täglichen Arbeitsabnahme: Datenverwaltung, Vereinheitlichung, Eingabeprüfung und Ähnliches. „Möglich“ beschäftigt sich mit neuen Umsetzungsideen, um realistischere, schnellere oder beeinflussbarere Ergebnisse zu erzielen.

Einfach und möglich

Ein typisches Anwendungsgebiet für die Datenverwaltung ist das Speichern und Laden. In einer Produktion ist kaum etwas so sicher wie diese Prozesse, genauso wie ihre Eigenschaft ein Motivationsfresser zu sein: Einhaltung von Pfad- und Namenskonventionen beim Speichern, Navigieren durch die Verzeichnisse beim Laden und vor allem der Umgang mit falsch benannten oder platzierten Dateien.

Eine einfache Applikation, die dem Nutzer dies innerhalb einer Pipeline-Konvention abnimmt, ist deswegen immer einer der ersten Ansätze, um den Arbeitsalltag zu vereinfachen. An diesem Punkt greifen vor allem Python und Py Side, insbesondere in der Kombination mit Maya, Nuke und Houdini, wie ein Zahnrad ins nächste. Durch ihre Integration genügt es, nur jeweils ein Skript zu entwickeln, das von allen drei Paketen genutzt wird. Der einzige Unterschied liegt in der Ausführung am Ende des Prozesses, an dem die jeweilige Software differenziert den Befehl zum Öffnen und Speichern der Dateien nutzen muss.

userSetup.py

```

1 #
2 # title      userSetup.py
3 #
4 # content    start point for Maya
5 #
6 # dependence set "PYTHONPATH=$SOFTWARE_PATH;$PYTHONPATH"
7 #
8 # author     Alexander Richter
9 # email      contact@richteralexander.com
10 #
11
12 import sys
13 import os
14
15 import maya.cmds as cmds
16
17 print ("\nWelcome " + os.getenv('username'))
18 print ("\nBREAKINGPOINT: System is setting ...\n")
19
20 cmds.evalDeferred("from scripts import maya_settings")
21 cmds.evalDeferred("from scripts import menu")
    
```

Bei Maya heißt die Initialisierungsschnittstelle userSetup. Ist der PYTHONPATH darauf gerichtet, wird diese beim Start ausgeführt und erlaubt einen Zugang zu der Software, um Starteinstellungen (maya_settings) zu setzen oder eigene Menüelemente (menu) hinzuzufügen.

init.py

```

1 #
2 # title      init.py
3 #
4 # content    startup script for nuke before menu
5 #
6 # dependence set "NUKE_SCRIPT_PATH=$SOFTWARE_PATH;$NUKE_PATH"
7 #
8 # author     Alexander Richter
9 # email      contact@richteralexander.com
10 #
11
12 import nuke
13
14 print ("\nWelcome " + os.getenv('username'))
15 print ("\nProjectname: System is setting ...\n")
16
17 #-----
18 # VARIABLES
19 #-----
20 FPS             = '24'
21 RESOLUTION     = '1920 1080 PROJECT'
22
23 #-----
24 # SETTINGS
25 #-----
26 nuke.knobDefault("Root.fps", FPS)
27 nuke.addFormat(RESOLUTION)
28 nuke.knobDefault("Root.format", "PROJECT")
    
```

Nuke trennt zwischen Init und Menu. Init wird vor Menu ausgeführt und ist vor allem für die Kommandozeilen- und Renderverarbeitung von Nuke wichtig. Im Gegenzug kann Init keine Menüs in Nuke hinzufügen, da diese erst zum Eingabezeitpunkt (Interactive Session) des Programms erstellt werden.

menu.py

```

1 # author     Alexander Richter
2 # email      contact@richteralexander.com
3 #
4 #-----
5 #
6 # content    Maya menu
7 #
8 # dependence set "PYTHONPATH=$SOFTWARE_PATH;$PYTHONPATH"
9 #
10 # author     Alexander Richter
11 # email      contact@richteralexander.com
12 #
13
14 import os
15 import maya.cmds as cmds
16
17 print ("\nWelcome " + os.getenv('username'))
18 print("\nSetup ArMenu")
19
20 #-----
21 # MENU
22 #-----
23 menu = cmds.menu("ArMenu", ht = 1, p = "MayaWindow", l = "ArMenu", to = 1, )
24
25 # Save & Load
26 cmds.menuItem(p = menu, l = 'save', c = 'from scripts import save;save.start()')
27 cmds.menuItem(d = True)
28
29 # SHD
30 cmds.menuItem(p = menu, l = 'SHD', os = True)
31
32 # ToolMenu
33 cmds.menuItem(p = toolMenu,
34              l = 'Combine SHD and MODEL',
35              c = 'from scripts.SHD import COMBINESHD_MODEL;COMBINESHD_MODEL.start()')
36
37 # HELP
38 cmds.menuItem(p = menu,
39              l = 'help',
40              c = 'import webbrowser;webbrowser.open("http://richteralexander.com/")')
    
```

menu.py ergänzt die Maya-Menüpalette mit dem eigenen Projektmenü, in dem Skripte mit Menü- und Untermenü-Einträgen verlinkt werden.

menu.py

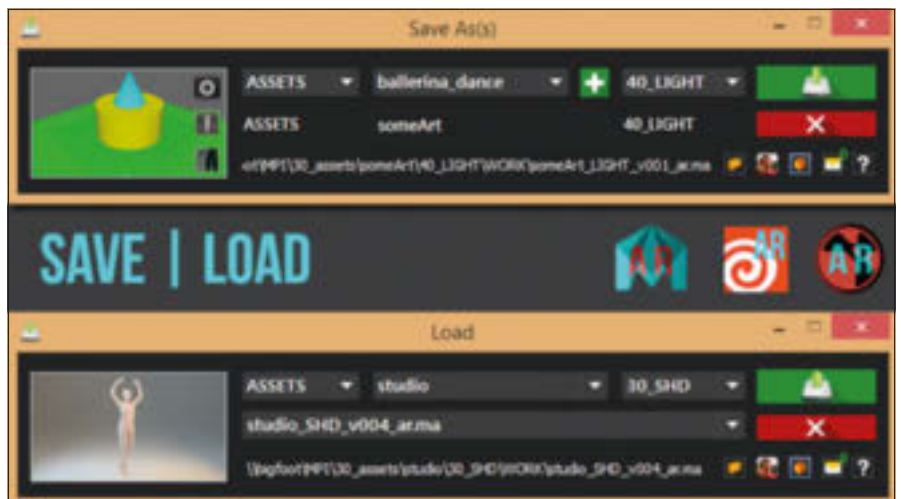
```

1 #
2 # dependence set "NUKE_MENU_PATH=$SOFTWARE_PATH;$NUKE_PATH"
3 #
4 # author     Alexander Richter
5 # email      contact@richteralexander.com
6 #
7 #-----
8 #
9 # content    Nuke menu
10 #
11 # dependence set "PYTHONPATH=$SOFTWARE_PATH;$PYTHONPATH"
12 #
13 # author     Alexander Richter
14 # email      contact@richteralexander.com
15 #
16
17 import os
18 import nuke
19 import webbrowser
20
21 print ("\nWelcome " + os.getenv('username'))
22 print ("\nPROJECT NAME: MENU")
23
24 #-----
25 # VARIABLES
26 #-----
27 PATH_SCRIPTS = ""
28 PATH_SCRIPTS = ""
29
30 #-----
31 # TOOLBAR
32 #-----
33 menuNode = nuke.menu("Nodes").addMenu("PROJECT NAME", icon = PATH_SCRIPTS)
34
35 menuNode.addCommand("save", lambda: save.start(), 'ctrl+alt+s')
36 menuNode.addCommand("load", lambda: arload.start(), 'alt+l')
37 menuNode.addCommand("erase", lambda: nuke.createNode(PATH_SCRIPTS), 'alt+w')
38 menuNode.addSeparator()
39 menuNode.addCommand("help", lambda: webbrowser.open("http://richteralexander.com/"), 'alt+h')
    
```

Für die Menüs ist – Welch Überraschung – die menu.py zuständig, in welcher Menüs, Toolbars, Nodes und weitere Schnittstellen definiert werden.

Es ist sogar möglich, noch einen Schritt weiter zu gehen und das Ladeskript in das Speicherskript zu integrieren, um die Arbeit des einen für das andere zu nutzen (siehe „Save and Load“). Damit wird nicht nur der Wartungsaufwand minimiert, sondern durch die Einheitlichkeit auch das Verständnis der Nutzer vereinfacht.

Save und Load ist in einem Skript zusammengefasst, das sich innerhalb von Maya, Houdini und Nuke gleichermaßen nutzen lässt. Gleichzeitig ist es möglich, dieses mit etwas Vorausblick betriebssystemunabhängig für Windows, Linux und Mac OS zu adaptieren.



Dropbox in der Pipeline



Schon oft wurden wir gefragt, wie man denn bei einer Teamgröße zwischen Einzelkämpfer und kleiner Truppe mit dedizierter Pipeline seine Projektdateien und Settings synchron hält – die Antwort darauf lautet oft: „Mit der Dropbox, natürlich!“ Die häufigsten Fragen dazu haben wir einfach Oliver Blüher, Country Manager DACH & Nordics von Dropbox, selbst gestellt.

DP: Einer der Punkte, der normale Filesyncs schwierig macht – insbesondere bei Metadaten, I/O-Sessions und versionierten Arbeitsdateien – ist die hohe Frequenz der Änderungen. Wie oft überprüft Dropbox die Dateien und synchronisiert?

Oliver Blüher: Dropbox syncnt automatisch und sofort: Sobald das System eine Dateiänderung sieht, wird diese hochgeladen, inklusive aller Änderungen.

DP: Kann man auch außerhalb des klassischen Dropbox-Folders Dateien synchronisieren?

Oliver Blüher: Es gibt eine Handvoll NAS, die von Herstellerseite aus bereits mit der Dropbox arbeiten, und einige User, die bereits Verknüpfungen zu SAN/NAS-Umgebungen hergestellt haben, um so eine Synchronisation zu bekommen. Meistens als symbolische Links oder Junction Points – obwohl wir das nicht wirklich empfehlen. Denn in dem Moment, wo der Link oder der Point nicht mehr verbunden sind, erkennt die Dropbox dies als Löschung und synchronisiert dementsprechend.

DP: Gibt es eine Infrastruktur bei der Dropbox, die sie „ansprechbar“ für TDs macht, zum Beispiel per API?

Oliver Blüher: Es gibt sogar zwei APIs: Die „Dropbox Core API“ (welche mit dem Account synchronisiert, Uploads und Dateitransfer handelt und auf den Inhalt der Ordner zugreifen kann) und die „Dropbox for Business API“ (mit der man User Accounts verwalten und in deren Namen Funktionen ausführen kann, zur Verwaltung von großen Teams). Die Dropbox Enterprise hat auch weitere Schnittstellen, unter anderem für uncapped API-Aufrufe. Und wenn man diese dokumentierte API beherrscht, kann man es in fast jedes System einhängen, welches das Team für sich ausgesucht hat.

DP: Kann man darauf auch ohne eigene Software-Entwicklung zugreifen?

Oliver Blüher: Da gibt es verschiedene Varianten. Wir haben zum einen ein ziemlich umfangreiches Partnerprogramm (siehe www.dropbox.com/business/app-integrations), bei dem wir direkt mit Softwareherstellern zusammenarbeiten. Diese Verknüpfungen und Schnittstellen sind meistens über die bereits erwähnte API gelöst. Viele der großen Partner haben dafür eigene Programmierer, die es ideal verknüpfen. Zum anderen greifen einige User zum Beispiel per Dropbox Desktop auf die Dateien der Creative Cloud zu, da diese in der Lage ist, wesentlich größere Dateien ungeachtet des Dateityps zu verwalten.

DP: Wo kann man die Dropbox in einer Multi-User-Pipeline einsetzen? Gibt es bereits Erfahrungswerte?

Oliver Blüher: Die Dropbox spricht mit Hunderttausenden von Anwendungen – über die Dropbox Business API auch mit einigen hochkarätigen Firmen. Im Animations/Filmworkflow speziell werden gerne einige der „Share“-Features verwendet. Viele Kunden greifen darauf zurück für Dateien, die normalerweise auf einem internen Server liegen würden. Zusätzlich verwenden einige Posthäuser die Sync-Engine und die Dropbox Video Preview für Dailies – sowohl zum Vorführen unterwegs als auch zum Hosten. Denn im Gegensatz zu anderen Systemen wie beispielsweise PIX oder DAX kann die Dropbox Preview auch sehr gut mit schwachen Internet-Verbindungen umgehen, zum Beispiel, wenn man unterwegs ist. Einen Feature-Wunsch, den wir auf dem Schirm haben, ist der klassische File Request, der für Postproduction-Studios und Filmemacher relevant ist, die mit vielen Vendors arbeiten.

Pfade und Funktionen

Der Pfad- und Funktionsaufbau ist ein wichtiges Element jedes Skripts. Pfade zu relativieren, sichert einem zwar Portabilität, doch bedeutet es noch lange nicht, dass diese flexibel sind.

Bei einer Änderung der Pfadstruktur oder Benennung muss diese in allen tangierten Skripten angepasst werden. Besser ist es, den Löwenanteil der Pfade aus einer Quelle zu beziehen und sie dort relativ zueinander zusammenzustecken. Ebenfalls empfiehlt es sich, Konstanten, wie die Namen der Tasks (Model, SHD etc.), zu definieren, um unkompliziert auf Ablaufanpassungen reagieren zu können.

Alternativen

Natürlich besitzen die meisten Unternehmen ihre eigene Pipeline. Leider wird diese zu meist unter Verschluss gehalten, da sie potenziell einen Produktionsvorteil birgt. Fraglich wäre, ob das angepasste System Dritten den gleichen Nutzen eröffnen würde.

Das Internet bietet zu diesem Thema versprengt Einzellösungen an. Open Pipeline ermöglicht eine einfache Datenverwaltung für Maya-Nutzer.

Shotgun und Ftrack kombinieren Zeiterfassung, Planung, Speicher- und Kommentarelemente, müssen aber erst einmal gezähmt werden und eignen sich erst ab einer gewissen Teamgröße und Einarbeitungsphase.

Fazit

Bill Gates sagte einst: „Ich würde immer einen faulen Menschen wählen, um einen schwierigen Job zu erledigen. Denn er würde sicherlich einen einfachen Weg finden, dies zu tun.“ Seid faul in diesem Sinne und nehmt euch die Zeit, den Zeitfresser anzugehen.

> ei



Alexander Richter ist Technical-Director-Student an der Filmakademie Baden-Württemberg, spezialisiert auf Shading, Lighting, Pipeline und Compositing.
www.richteralexander.com

Links

arPipeline – Nuke
▷ vimeo.com/148166022
(Videos zu Basics, Maya und Houdini folgen)

arPipeline – Initialisierungsskripte (Download)
▷ richteralexander.com/portfolio/arPipeline.html

Da Windows, Linux und Mac OS ebenfalls diesen Dreivölkerbund mitsamt unseren Helfern (Python und Py Side) nutzen, lohnt es, sich über die Betriebssystemunabhängigkeit der

Skripte Gedanken zu machen. Ihre Lauffähigkeit hängt meist von Systemvariablen, differenzierten Pfadaufbauten, Back/Forward-Slashes und der Modalität des Systems ab.