2022
6
ISSN 1433-2620 > 26. Jahrgang >> www.digitalproduction.com
Publiziert von Busch Glatz Germany GmbH

Deutschland € 17,90
Österreich € 19,–
Schweiz sfr 23,–

DIGITAL PRODUCTION

DIGITAL PRODUCTION

MAGAZIN FÜR DIGITALE MEDIENPRODUKTION

NOVEMBER | DEZEMBER 06:2022

ACES!
Für jede Pipeline in allen Tools

Projekte
Strange World, Arcane, Love, Death & Robots . . .

Tools
C4D 2023, Nuke, After Effects, Obsbot, Gyros

Interviews
Riverside, Hammerspace, Marquis, Solaris und mehr!

Nuke Tools Vol.1 – Script Utilities

Beginnend mit dieser Ausgabe, wollen wir eine neue Serie in der DP lostreten, die hoffentlich allen Nuke Userinnen gleichermaßen entgegenkommt: Nuke erlaubt es seiner Nutzergemeinde, dank Python, Blinkscript und Co. sehr einfach eigene Tools zu kreieren und der Community zur Verfügung zu stellen. Hier den Durchblick zu behalten ist allerdings nicht ganz so einfach.

von Christoph Zapletal

Wer auf **nukepedia.com** schaut, findet zwar schon eine gute Unterteilung in Kategorien, aber selbst in jeder dieser Kategorien gibt es dutzende, teilweise hunderte Gizmos oder Scripte. Eine Riesenerleichterung für viele Artists war dann letztes Jahr die Veröffentlichung des Nuke Survival Toolkits von Tony Lions. Endlich eine Sammlung von Tools, kuratiert und dokumentiert. Und trotzdem werden viele, die das Toolkit installiert haben, feststellen, dass sie nur einen Bruchteil der Tools nutzen oder überhaupt wissen, was die Tools genau machen und wie sie angewendet werden. Denn seien wir mal ehrlich: Man sucht nach einem Tool, wenn man es braucht – nicht vorher. Hier wollen wir ein bisschen früher ansetzen – und Euch zeigen, wie Ihr mit den richtigen Tools noch mehr aus Nuke herausholen könnt.

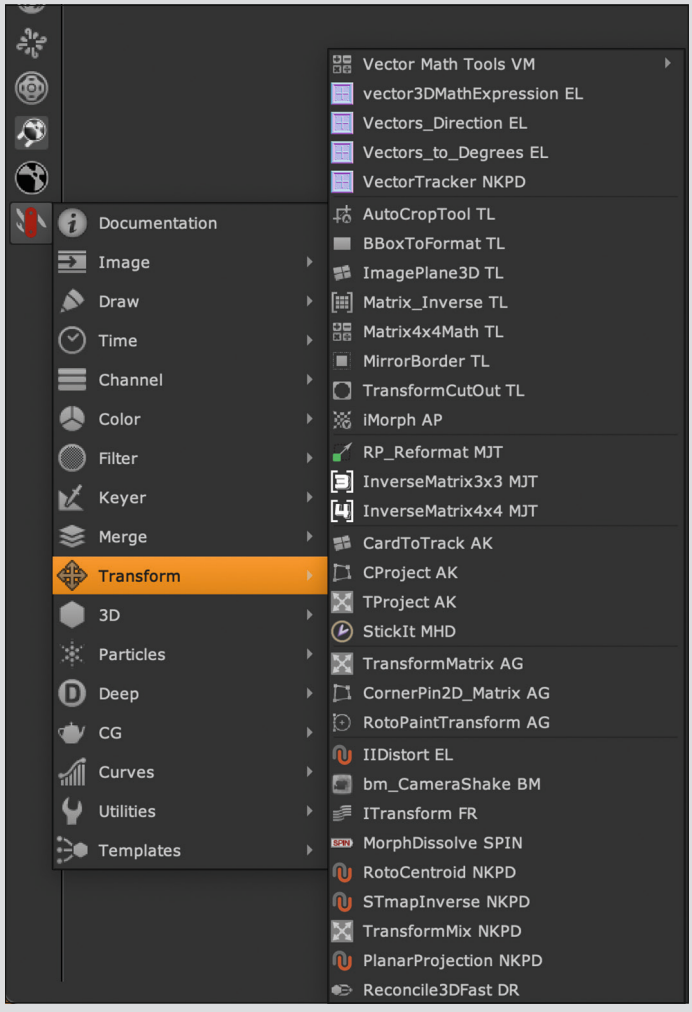
Der Schwerpunkt für diese Ausgabe sind Script Utilities, also alles, was Euch beim Erstellen und Managen eurer Scripte hilft. Und natürlich haben wir sichergestellt, dass alle hier vorgestellten Tools auch unter Python 3 und damit in Nuke 13 laufen. Bevor wir hier aber in medias res gehen, wollen wir einen kleinen Auffrischungskurs machen – was sind Toolsets und Gizmos, wo werden sie installiert und wie stellt man sicher, dass sie auch ein Update überstehen.

Toolsets vs. Groups vs. Gizmos

Der einfachste Weg, sich ein eigenes Tool in Nuke zu bauen, ist das Erstellen eines Toolsets. Mehrere Nodes markieren, auf den Schraubenschlüssel an der Seite klicken und „Create" auswählen. Was jetzt im Hintergrund passiert, ist Folgendes: In einem Subfolder eures .nuke-Folders wird ein eigenes Nuke Script eurer Auswahl abgespeichert und jedes Mal, wenn Ihr dieses Toolset aufruft, wird dieses Script in das offene Script importiert. Wer aber nicht jedes Mal zig verschiedene Nodes laden möchte, sondern eine kleine, übersichtliche Node haben möchte, für den sind dann Groups der nächste Schritt: Wieder die relevanten Nodes markieren, dann „Collapse to Group" auswählen, noch ein paar Slider und Knobs dazu, fertig. Dies hat den großen Vorteil, dass man nach wie vor auf die einzelnen Nodes, aus denen ein Tool besteht, zugreifen und sie modifizieren kann.

Der große Nachteil ist, dass Groups genau so in der Script-Datei abgelegt werden wie die Nodes, aus denen sie bestehen. Warum ist dies ein Nachteil? Nun, zum einen kann ein Script so sehr leicht aufgebläht werden, und ab gewissen Größen wirkt sich dies nun mal auch auf die Performance aus. Zum anderen lässt sich eine Group, ist sie einmal in einem Script angewendet worden, nicht mehr updaten. Für einen einzelnen Artist ist dieses vielleicht nicht ganz so relevant, aber sobald innerhalb einer Pipeline solche individuellen Tools ausgetauscht werden sollen, beginnt mit Groups der Albtraum eines jeden TDs – hier sind Gizmos eine bessere Lösung.

Ein Gizmo ist ein aus einer Group heraus erstelltes Plug-in für Nuke, was jetzt eben nicht mehr in seiner Gänze im Script abgespeichert wird, sondern nur noch – wie jedes andere Plug-in eben auch – referenziert wird. So kann ein Gizmo dann auch leicht in einem laufenden Projekt geupdated werden. Deswegen werden wir in den allermeisten Fällen in dieser Reihe über Gizmos sprechen. Zu Beachten sind bei Gizmos allerdings zwei Dinge: Zum Einen lassen sich – anders als bei den Groups – die einzelnen Nodes nicht mehr editieren. Zum anderen müssen sie, sofern eine Renderfarm genutzt wird, auch auf allen Clients installiert werden.



Das Nuke Survival Toolkit ... eine Installation, hundert neue Tools

Installieren – aber richtig!

Zuallererst: Diese Anleitung ist für Nuker gedacht, die auf Ihrer eigenen Workstation Gizmos installieren wollen. An alle, die ein Gizmo in einer Pipeline installieren wollen: Please ask your friendly Neighbourhood TD!

Wie zuvor erwähnt ist es in Pipelines, gerade in Blick auf die Renderfarm, aber natürlich auch in Hinblick auf das Wechseln von Artists und Workstations, extrem wichtig, dass Gizmos zentral verwaltet werden. Und dies liegt – glücklicherweise – in den Händen der Pipeline TDs und nicht der Artists.

Um ein Gizmo zu installieren, reicht es theoretisch, wenn es im .nuke-Folder der jeweiligen Workstation liegt. Dieser befindet sich auf Windows unter Drive letter:\ Documents and Settings\login name\.nuke, auf Linux unter /users/login name/.nuke und auf Mac OS unter /Users/login name/ .nuke. Jedes hier oder in einem Subfolder abgelegte Gizmo wird von Nuke geladen. Man würde jetzt davon ausgehen, das damit die Installation beendet sei – nun, falsch gedacht, denn Nukes Möglichkeit, per Python an quasi jede Pipeline angepasst zu werden, fällt uns jetzt gehörig auf die Füße. Die Gizmos sind zwar da und können über ein TCL Kommando auch aufgerufen und in den Node Tree geladen werden, in die UI, also Side Bar, Search Box und Context Menu sind sie allerdings noch nicht integriert. Dafür müssen wir ein kleines bisschen Python anwenden.

Genauer gesagt wollen wir Einträge in zwei Python-Scripten vornehmen: Der menu.py und der init.py. Wenn die Nuke-Installation noch „Vanilla", also ohne irgendwelche hinzugefügten Gizmos ist, dann wird es diese Files noch nicht geben und wir können sie mit einem Texteditor erstellen. Wichtig ist nur, dass man drauf achtet, dass diese Files im .nuke-Folder liegen und keinen zusätzlichen Suffix außer „.py" haben.

Fangen wir an mit der menu.py, welche Nuke sagen wird, welche Gizmos wir in unser Menü, also in die Seitenleiste und damit auch ins Context Menu und die Search Box, integrieren wollen. Hier beginnen wir mit einer Variable, die genau definiert, was alles unter diesem neuen Menüpunkt gesammelt wird. Sie lautet:

```
toolbar = nuke.menu("Nodes").
addMenu("MyTools")
```

MyTools ist in diesem Beispiel natürlich nur ein Platzhalter und kann individuell modifiziert werden. In der nächsten Zeile wird die von uns gerade definierte Variable genutzt, um unser Gizmo hinzuzufügen.

```
Toolbar.addCommand("MyGizmo",
"nuke.CreateNode("MyGizmo")")
```

Auch wieder hier ist "MyGizmo" ein Platzhalter, in den natürlich der Name des Gizmos ohne Extension mit korrekter Groß- und Kleinschreibung einzutragen ist. Wenn wir das File nun so abspeichern und Nuke neu starten, sollten wir mit einem neuen Symbol im Toolbar begrüßt werden und dort das von uns installierte Gizmo nun auch in der UI sichtbar sein.

Das ist alles ganz praktikabel, wenn man mit ein oder zwei Gizmos arbeiten will, aber wir wollen ja im Lauf der Zeit einiges mehr installieren und da wäre es ja ganz ratsam, wenn wir unsere Gizmos auch in einen oder mehrere Unterordner unseres .nuke-Folders wegsortieren könnten. Im Moment würden die aber von Nuke ignoriert werden und in dem Moment, wo wir unsere Gizmos in einen solchen Subfolder verschieben, würde Nuke die Gizmos nicht mehr finden. Abhilfe wird hier die init.py schaffen, die wir mit dem Eintrag

```
nuke.pluginAddPath('MyGizmos')
```

füllen, wobei „MyGizmos" natürlich der Platzhalter für den Namen des erstellten Subfolders ist. Jetzt wird dieser bei jeder Initialisierung von Nuke dem zu lesenden Inhalt hinzugefügt und so kann dann in der Folge die menu.py auch die in ihr enthaltenen Gizmos finden. Natürlich kann die menu.py um beliebig viele Einträge erweitert werden, Nuke kann jetzt also noch ordentlich aufgepimpt werden.

Und Python?

Was Gizmos für die optische Veränderung von Pixeln ist, sind Python Scripts für alles Organisatorische in unserem Node Tree. Um sie zu starten, kann man natürlich den Python Editor von Nuke benutzen. Oder man installiert das Script anständig, dann hat man es ganz bequem oben in der Menüleiste. Und das Verfahren ist auch dem für Gizmos ganz ähnlich. Zuallererst erstellen wir im .nuke-Folder einen „Python"-Subfolder und fügen unserer init.py noch eine Zeile hinzu, um diesen beim Start von Nuke auch mit einzubeziehen:

```
nuke.pluginAddPath('Python')
```

In der menu.py müssen wir nun auch wieder wie bei den Gizmos eine Variable setzen, wobei diese jetzt nicht für die Nodes gilt, sondern für die Menüleiste:

```
PYtoolbar = nuke.menu("Nuke").
addmenu("MyTools")
```

Jetzt müssen wir noch die entsprechenden Python-Scripte in Nuke importieren, was wir mit folgendem Eintrag in die menu.py erreichen:

```
Import Myscript
```

„Myscript" ist hier ein Platzhalter für den Namen des zu importierenden Scripts. Und jetzt müssen wir noch dafür sorgen, dass in unserem neuen Menüreiter „MyTools" auch ein Menüpunkt mit unserem Script erscheint und dieses auch ausgeführt wird, wenn wir draufklicken

```
PYtoolbar.addCommand("MyScript",
"MyScript.function()")
```

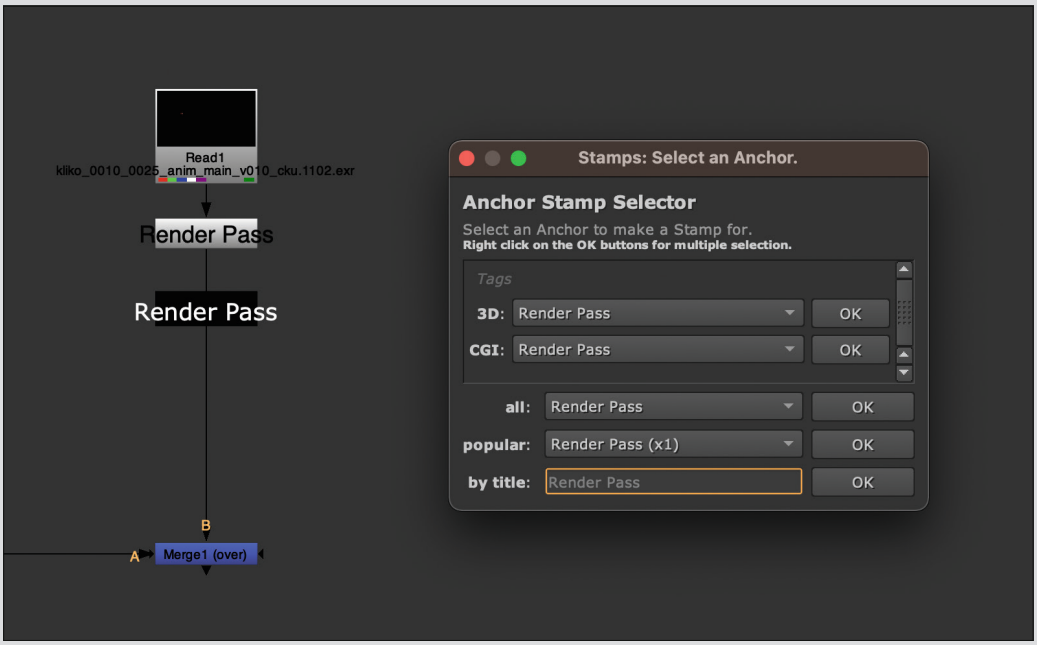
Der erste Eintrag in Anführungszeichen steht hier tatsächlich für die Schreibweise des Eintrages im Menüreiter, kann also auch noch beliebig angepasst werden. Das zweite „MyScript" allerdings bezieht sich auf den Namen des Scriptes. Hier ist eine korrekte Schreibweise also extrem wichtig. Das „.function" bezeichnet die Funktion, die innerhalb des Scriptes aufgerufen werden muss.

Wem das jetzt alles etwas zu schnell geht – keine Sorge: In der Regel liegen jedem Gizmo und jedem Script Readme-Files oder Beispiel menu.py und init.py Files bei, wo man die entsprechenden Zeilen relativ schnell mit Copy-Paste hinzufügen kann. Aber diese kleine Anleitung hat hoffentlich dazu geführt, das man jetzt auch weiß, was man da hin- und herkopiert.

Stamps

Adrian Pueyo und Alexey Kuchinski haben mit Stamps ein Tool geschrieben, was auf den ersten Blick eine verschlankte Version der nativen Postage Stamp von Nuke ist, was aber so viel mächtiger und effizienter ist, dass es sich mittlerweile in mehreren Studios als Standardtool in der Pipeline durchgesetzt hat.

Die Idee von Stamps ist relativ simpel: An jeder Stelle im Nodegraph kann ein sogenannter „Anchor" platziert werden. Von diesem Anchor aus können dann beliebig viele mit ihm verbundene Stamps erstellt und im Script verteilt werden. Dabei sind die Verbindungen zwischen Anchor und Stamps nur sichtbar, wenn man eine Stamp selektiert hat. Das gestaltet das Script sehr viel übersichtlicher. Bis hierhin entspricht das Verhalten dem der Postage Stamps. Allerdings fallen schon früh beim Benutzen große Unterschiede auf. Zum Einen erbt ein Stamp automatisch den Namen des Anchors und auch wenn viele Stamps im Script existieren, können sie den gleichen Namen tragen, ohne dass dieses zu Konflikten im Script führt. Dies liegt daran, dass Stamps für den Artist unsichtbar jedem Anchor einen individuellen „Hash" vergibt, also einen einzigartigen Identifier. Zum anderen kann man – anders als bei der Postage Stamp von Nuke – einen Stamp kopieren und die Verbindung zum Anchor bleibt bestehen.



Gegen Stamps sehen die Nuke-eigenen Postage Stamps ziemlich blass aus.

Aber Stamps kann noch mehr. Es verfügt über ein Tagging-System. Schon beim Erstellen eines Stamps kann man beliebig viele Tags hinzufügen, mit deren Hilfe man sich am anderen Ende des Scriptes schnell einen neuen Stamp generieren kann, auch, wenn der dazugehörige Anchor mehrere dutzend Mausklicks weit entfernt liegt. Um all diese Funktionen trotzdem übersichtlich zu halten, nutzt Stamps einen Universalhotkey: F8. Hat man Footage oder eine Node selektiert und drückt man F8, wird sofort ein Anchor mit dazugehörigem Stamp generiert und ein PopUp-Menü erlaubt die Vergabe von Namen und Tags. Ist nichts ausgewählt, kann man mit F8 ein Menü aufrufen, über das man sich einen neuen Stamp aus einem bereits im Script existierenden Anchor erstellen kann. Dabei kann man aussuchen, ob man dieses auf Basis der Tags, des verwendeten Backdrops oder aber auf Grund der Beliebtheit eines gewissen Anchors macht. Das heißt, je öfter ein Anchor von Stamps genutzt wird, desto weiter oben ist er in der Auswahlliste.

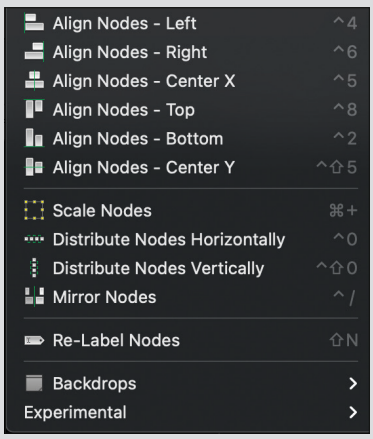
Eine besondere Verbesserung gegenüber den Postage Stamps ist übrigens, dass diese Stamps mit allen Arten von Nodes zurechtkommen, also auch zum Beispiel einer Camera oder Deep Passes. Und um die Kompatibilität muss man sich auch nicht sorgen – selbst auf Workstations, auf denen Stamps nicht installiert ist, lassen sich die Skripte problemlos lesen und ändern. Einzig und allein Neue Stamps zu erstellen und die Reconnect- und Suchoptionen stehen einem dann nicht zur Verfügung.

Zu guter Letzt sei noch erwähnt, dass Stamps auch beim Wiederverknüpfen sehr flexibel ist. Zuallererst wird es immer versuchen, seinen Anchor auf Basis des individuellen Hashs zu finden. Es gibt aber auch die Möglichkeit, den Namen des Anchors als Basis zu nehmen. Und das eröffnet ganz neue Möglichkeiten, wenn man Stamps nutzt, um sich ein Toolset zu erstellen, was mehrere Artists nutzen sollen. Sofern sich an eine gewisse Nomenklatur gehalten wird, sei es zum Beispiel „Plate" für das Footage und „Cam" für die Kamera, wird sich ein Stamp in einem geladenen Toolset automatisch mit dem entsprechenden Anchor verbinden, wenn die Option „Reconnect by name" aktiviert ist.

bit.ly/nuke_stamps

Nuke Nodegraph Utils

Die Nuke Nodegraph Utils sind eigentlich gleich mehrere Tools, die Erwan Leroy hier in einer kleinen Sammlung von Python Scripts zusammengefasst hat. Gemein ist ihnen allen, dass sie Arbeitswege verkürzen und Klicks einsparen. So findet man nach der Installation in der Menüleiste mehrere Optionen zum Ausrichten zuvor selektierter Nodes. Die richtigen Zeitsparer finden sich aber in der „Scale Nodes"-Option, dem „AutoBackdrop"-Script und der „Re-Label Nodes" Option. Wird „Scale Nodes" auf einer Auswahl von Nodes aktiviert, erscheint um die Nodes ein Rechteck mit Handles, ähnlich einer Editbox um einen Spline. Verändert man jetzt diese Handles, werden nicht die Nodes selber größer, aber die Abstände zwischen den Nodes werden proportional größer. Dabei bietet das Tool auch die Option, die anhängenden Teile des Scripts mitzuskalieren. So kann leicht Platz für zusätzlich Nodes geschaffen werden. „Auto Backdrop" ist vor allem dafür nützlich, dass es dem Hinzufügen von Backdrops endlich einen Shortcut zuweist und so diese Aufgabe sehr viel effizienter gestaltet. Nodes auswählen, „Alt+B" drücken, im Pop-Up Dialog einen Namen vergeben und fertig. Und in genau die gleiche Kerbe schlägt die „Re-Label Nodes"-Option, mit der man sich zum anständigen Bearbeiten des Labels seiner Nodes endlich den Klick in die Properties sparen kann. Node auswählen, „Shift-N" drücken und das Label vergeben. Es gibt dann noch ein paar Funktionen, die vom Autor selbst als „Experimental" bezeichnet werden und das Verbinden und Trennen von Connections im Node Tree per Draw im Stile von Houdini ermöglicht. **bit.ly/nuke_nnu**

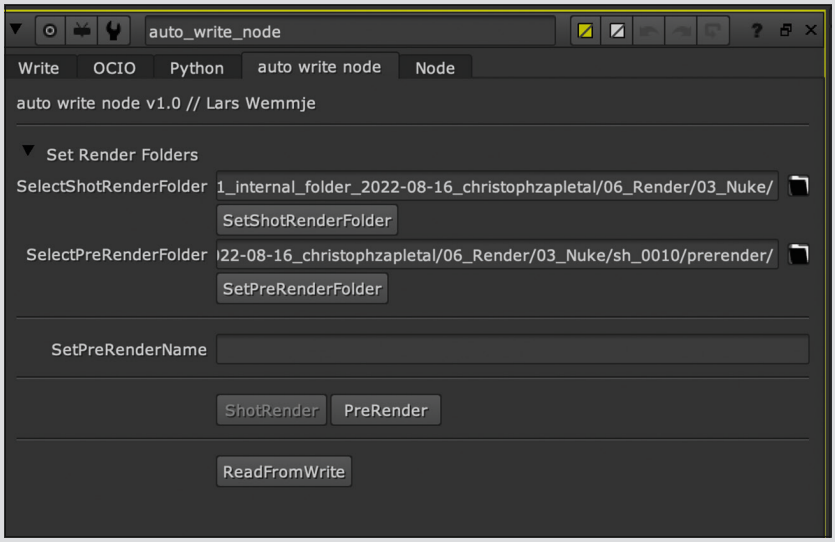


Zu wenig Platz für noch mehr Nodes? Diese Tools können da helfen ...

Auto Write Node

Die Auto Write Node richtet sich explizit an Artists, die nicht in einer großen Pipeline mit Renderfarm arbeiten, sondern die auf Ihrer eigenen Workstation ein bisschen mehr Automation suchen. Die Node – einmal eingerichtet – übernimmt komplett die Benennung und Versionierung der zu rendernden Files, und das in jeder beliebigen Ordner-Strukter. Selbstverständlich muss dafür immer mit derselben Ordnerstruktur gearbeitet werden. In einer Hinsicht kommt dieses Tool etwas anders daher als die anderen hier vorgestellten Helferlein. Die Auto Write Node wird weder als Gizmo noch als Python Script, sondern vielmehr als kleines Nuke Script ausgeliefert, welches man sich dann in ein individuelles Toolset umwandelt, um die Auto Write Node in jedem Script verfügbar zu haben.

Zum Einrichten erstellt man sich am besten ein Dummy-Projekt mit der zugrunde liegenden Ordnerstruktur. Dann erstellt man an dem Ort, wo üblicherweise die Nuke Scripte abgelegt werden, ein leeres Nuke Script und benennt dieses genauso, wie man es sonst üblicherweise in einem üblichen Projekt machen würde. Nun importiert man sich die runtergeladene Auto Write Node und setzt Pfade zu den jeweiligen Speicherorten für Render und Prerender. Dabei kann die zugrunde liegende Ordnerstruktur Shot- oder Taskbased sein. Innerhalb der Auto Write Node werden nun relative Pfade zu den jeweiligen Speicherorten gesetzt. Legt man diese Node nun als Toolset ab, ist sie in jedem zukünftigen Nuke Projekt als Auto Write Node verfügbar.

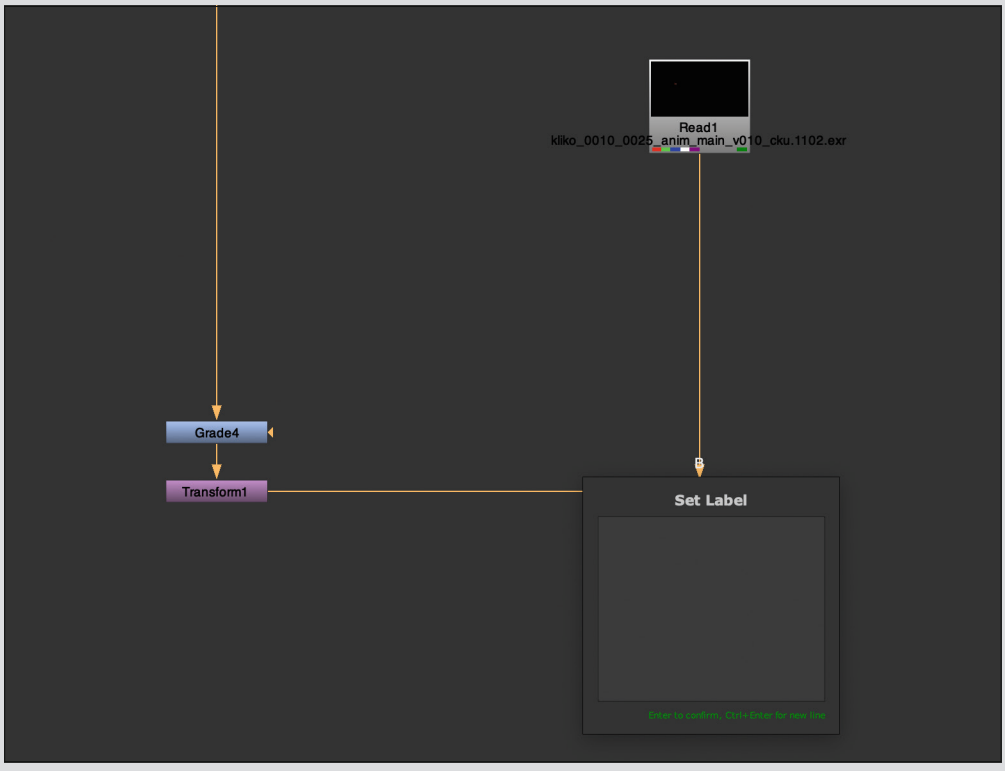


Ganz ohne Pythonkenntnisse kann man über dieses Menü alle relevanten Pfade definieren.

Autor Lars Wemmje, vielen vielleicht bekannt durch seinen Youtube-Kanal „Love VFX" hat hier ein wirklich hilfreiches Tool erstellt, welches dem Artist nicht nur eine Menge Tipperei erspart, sondern auch die Fehlerquote durch falsch benannte Files massiv reduziert. **bit.ly/nuke_awn**

Auto Project Settings

All diejenigen unter uns, die nicht mit einer ShotGrid-Subscription und einer ordentlichen Pipeline gesegnet sind, kennen das: Neues Script anlegen, Footage laden. Dann ab in die Scene Settings und die Frame Rate festlegen. Anfangen zu arbeiten und sich nach fünfzehn Minuten ärgern, dass man die Scene Resolution nicht gleich mit eingestellt hat. Dann das erste Mal das Script speichern



Anstatt sich durchs Properties Panel zu klicken, hat man jetzt ein elegantes Popup für seine Labels.

und feststellen, dass man sich noch keine Ordnerstruktur angelegt hat. Weiterarbeiten. Dann das erste Rendering abfeuern, also Write Node erstellen und dreimal den Dialog wieder schließen, um nach der korrekten Benennung zu schauen. Rendern. Sich ärgern, dass man ein Standbild gerendert hat, weil der Source Clip nicht den richtigen Timeoffset hat. So langsam seine beruflichen Entscheidungen überdenken …

Luciano Cequinel hat mit „Auto Project Settings" ein kleines Python Script erstellt, was dem Artist einige der oben beschriebenen Frustrationen ersparen sollte. Denn wenn man es sich recht überlegt, sind alle entscheidenden Infos, die man braucht, im Footage. Deswegen ist es auch nur folgerichtig, dass nach der Installation des Scripts in Nuke unter „File" die neue Option „New Comp from Footage" erscheint. In der nun erscheinenden Dialogbox kann man nun erst einmal das Footage auswählen, welches der Szene zu Grunde liegen soll.

Als nächstes kann man entscheiden, ob Nuke das Comp standardmäßig auf Frame 1001 anlegen soll. Wird diese Option gewählt, wird eine Time Offset Node eingefügt, die das Footage entsprechend verschiebt – ansonsten nimmt Nuke den normalen Frame Count als Grundlage.

Die zweite Option, „Export at Orignal Frame Range" sorgt dann im Gegenzug dafür, dass eine zweite TimeOffset Node das Comp wieder entsprechend verschiebt, damit die ursprüngliche Range exportiert wird. Das ist extrem praktisch, um für Animationen und Timings das Comp auf eine vernünftige Range zu schieben und am Ende doch sicherzustellen, dass die Clips exakt so rausgerendert werden, wie sie reingekommen werden.

Mit der Checkbox „Localization Policy" kann die vom Script erstellte Read Node gleich zum Cachen gebracht werden und mit „Use TCL Command on Write Node" kann man den Namen des Scriptes automatisch an die Write Node schicken.

Wer mag, kann an dieses Script auch noch ein Stück weit sein File Management auslagern. Entscheidet man sich, einen „Master Folder" festzulegen und den Scriptnamen bereits im Dialog festzulegen, kann man im letzten Schritt auch noch Subfolder eingeben, die das Script dann innerhalb des Masterfolder anlegt. Hierbei wird der erste Subfolder gleichzeitig als derjenige genommen, in dem das Script abgespeichert wird. Sofern man innerhalb einer shotbasierten Filestruktur arbeitet, kann das recht hilfreich sein, die Flexibilität der Auto Write Node bietet es aber nicht.

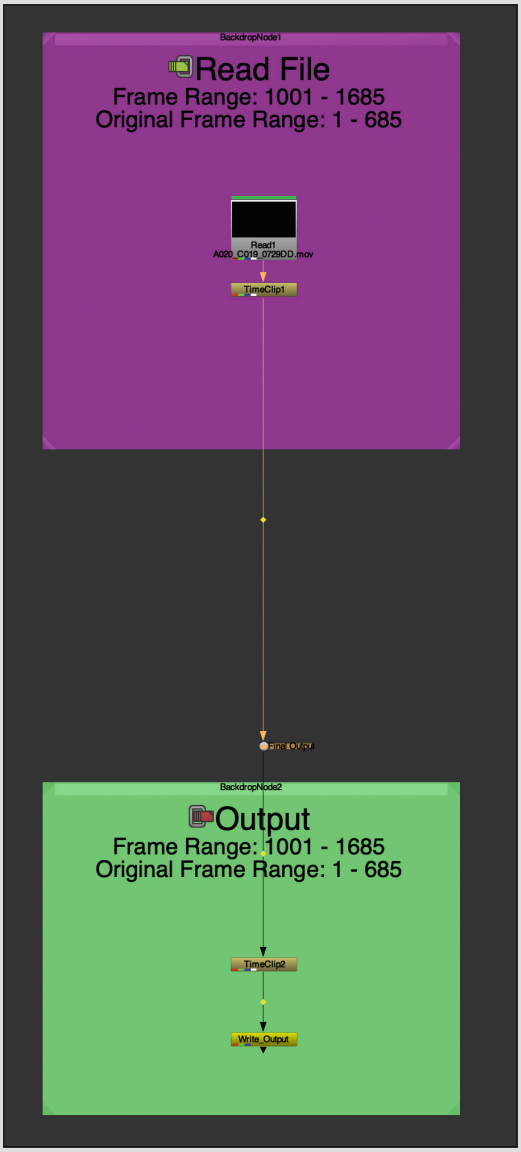
bit.ly/nuke_aps

Fazit und Ausblick

Die hier vorgestellten Tools sind wirklich nur eine kleine Auswahl des recht umfangreichen Angebots, welches es in der Nuke Community und insbesondere auf Nukepedia gibt. Doch alle haben meinen Workflow nachhaltig optimiert und beschleunigt. In diesem Sinne wird diese Serie in den nächsten Ausgaben der Digital Production fortgesetzt werden. In der nächsten Ausgabe werden wir uns all denen kleinen und großen Tools widmen, die Nukes User Interface pimpen und da geht es um wesentlich mehr als nur kosmetische Änderungen. **› ei**

Christoph Zapletal ist seit mehreren Jahren als freiberuflicher Compositing Artist und VFX-Supervisor in Hamburg tätig. Sowohl mit Autodesk Flame als auch The Foundrys Nuke arbeitete er bereits an diversen Werbe- und Spielfilmprojekten mit. Darüber hinaus ist er als Dozent für fxphd und an der HFF München tätig.
Mehr unter: **www.christophzapletal.de**

Mit so einem schicken Tree darf man gleich ins compen starten.



Einmal alle relevanten Fragen beantworten und sich hinterher nicht mehr ärgern müssen.