2017
5
DIGITAL PRODUCTION

ISSN 1433-2620 > B 43362 >> 21. Jahrgang >>> www.digitalproduction.com
Publiziert von DETAIL Business Information GmbH

Deutschland € 17,70
Österreich € 19,–
Schweiz sfr 23,–

DIGITAL PRODUCTION

MAGAZIN FÜR DIGITALE MEDIENPRODUKTION

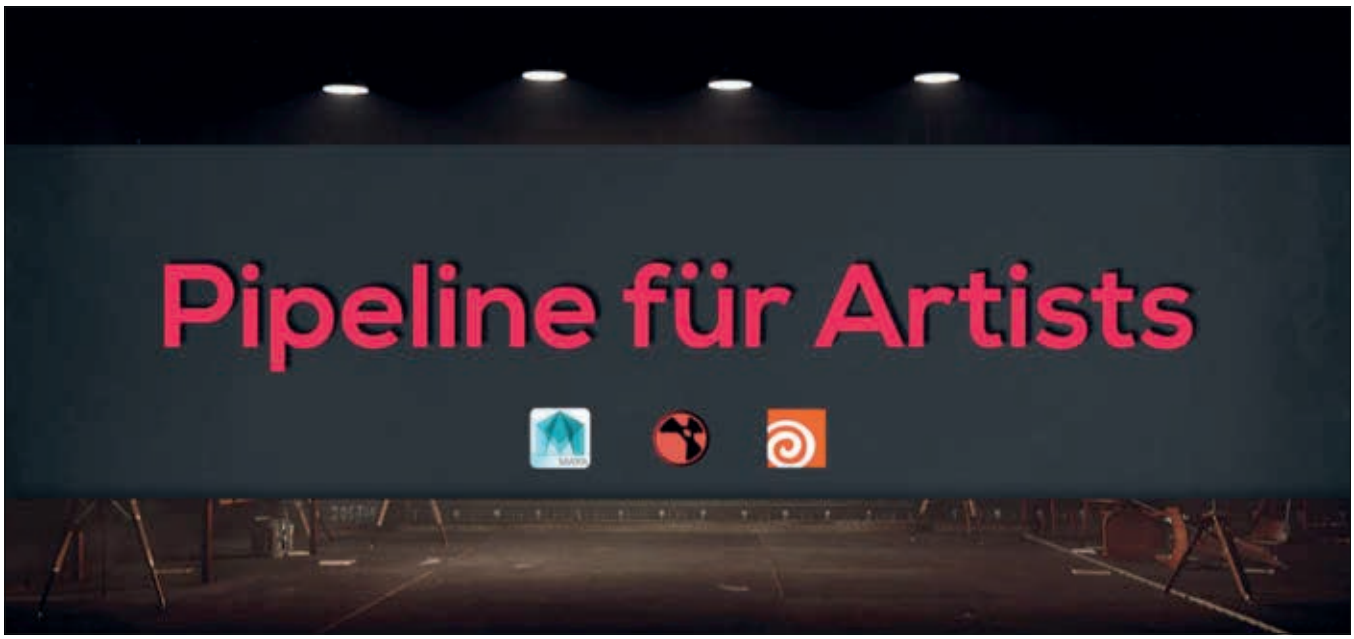
SEPTEMBER | OKTOBER 05:2017

Pipeline
Houdini, Fabric Engine und Scripting in neuem Licht

Förderung
Filmidee? Check! Auf in die Förderung – wir zeigen wie.

Tools & Tricks!
C4D R19, Arnold, Resolve 14, ZBrush 4 R8 & mehr

4 194336 217709 05 >



Nuke- und Maya-Tricks für den Alltag

Vereinfachen Sie sich den Alltag durch ein paar Kniffe und sparen Sie den täglichen Frust und die Zeitverschwendung für Prozesse, die zwar gemacht werden müssen, aber weder wirklich Spaß machen, noch Einfluss auf das fertige Bild haben.

von Alexander Richter

Im Alltag gibt es viele Momente unkreativer Arbeit. Sachen, die getan werden müssen, manchmal ohne, dass wir sie überhaupt zur Kenntnis nehmen. Über die Jahre haben wir uns angewöhnt, wie wir bestimmte Aufgaben erledigen: Ob Animation, Modeling oder Compositing. Wir öffnen bestimmte Menüs, navigieren zu bestimmten Ordnern, aktivieren Ketten von Befehlen oder warten auf deren Fertigstellung.

Prozesse sind uns wichtig, sie geben uns Sicherheit bei der Erfüllung der täglichen Arbeit oder der Umsetzung unserer kreativen Ideen. Auf der anderen Seite neigen wir dazu, bestimmte Automatismen als objektive Wahrheiten anzusehen, als einzige Wege, um etwas zu erreichen, ohne zu sehen, dass es mit ein bisschen Vorbereitung einfacher gehen kann.

Studios plagen dieselben Probleme, aber sie schaffen sich Abhilfe durch Pipelinesysteme, die so viel wie möglich von dieser „idle time" automatisieren, um die Produktivität der Artists zu erhöhen. Schaut man jedoch auf die Realität, wird die meiste 2D- und 3D-Arbeit von Freelancern, kleinen Gruppen, Privatpersonen oder Studenten gemacht, welche nicht von solchen Systemen profitieren und oft nicht wissen, wie sie ihren Arbeitsalltag vereinfachen können. Aus diesem Grund will ich in diesem Artikel auf ein paar offensichtliche und weniger offensichtliche Methoden eingehen.

Referenzen

Recherche ist nicht nur beim Erstellen eines Films wichtig, sondern auch, um ein Projekt planungstechnisch zu realisieren. Bevor man anfängt, professionell oder privat, in einer Gruppe oder alleine, sollte man sich für einen Moment hinsetzen und schauen, wie es die Großen machen. Oft gibt es Videos oder Interviews, wo erklärt wird, wie bestimmte Projekte angegangen oder knifflige Probleme gelöst wurden. Für manche können sich Bereiche wie Photogrammmetrie erschließen, während andere von Plug-ins oder Softwarepaketen erfahren.

Konventionen

Konventionen oder Regeln klingen erstmal ziemlich unkreativ, nehmen einem jedoch sehr schnell das Denken ab, vor allem bei den unkreativen und langweiligen Aufgaben. Der Moment, in dem man sich nicht mehr Fragen muss, wo man das aktuelle Rendering ablegen sollte, wie die Datei heißen muss oder wie die Szene aufgebaut werden sollte, ist der, wo man seinen Kopf frei macht für die Sachen, die einem wirklich wichtig sind: die Ideenumsetzung.

Weiterführende Informationen finden Sie im Artikel „Pipelines für Animations- und VFX-Produktionen" (**DP 02:2016, S. 8-12**) oder in meiner Open-Source-Pipeline-Dokumentation von Plex, wo es um Folder Structure (Ordnerstrukturen), Naming Conventions und Software Pipelines geht.

Templates

Bauen Sie sich Vorlagen. Das Wissen und die Erfahrungen, die Sie jeden Tag sammeln, können Sie bündeln, indem Sie vorgefertigte Szene erstellen, die Sie immer wieder als Vorlage nutzen.

Von Lightrigs, Shadern, Texturen bis zu ganzen Compositingbäumen ist alles möglich. Hauptsache, Sie bauen die Szenen so auf, dass nur ein Minimum an Handgriffen nötig ist, um sie für den aktuellen Fall anzupassen.

```
# Open folders and links
import webbrowser
path = r'C:\project'
path = r'http://richteralexander.com'
webbrowser.open(path)
```

Abb. 01: Öffnen von Pfaden und Links

```
# create screenshot (windows)
from PySide import QtGui
path = r'D:\screenshot.png'
QtGui.QPixmap.grabWindow(QtGui.QApplication.desktop().winId()).save(path, path.split(".")[-1])
```

Abb. 03: Erstellen von Screenshots

```
# OPEN current scene path
import webbrowser

# Maya **********************
import pymel.core as pm
path = pm.saveScene()
# ***************************

# Nuke **********************
import nuke
path = nuke.root()
# ***************************

path = os.path.dirname(path)
webbrowser.open(path)
```

Abb. 02: Codezeilen für die Erstellung der Speicherpfadordner (before render)

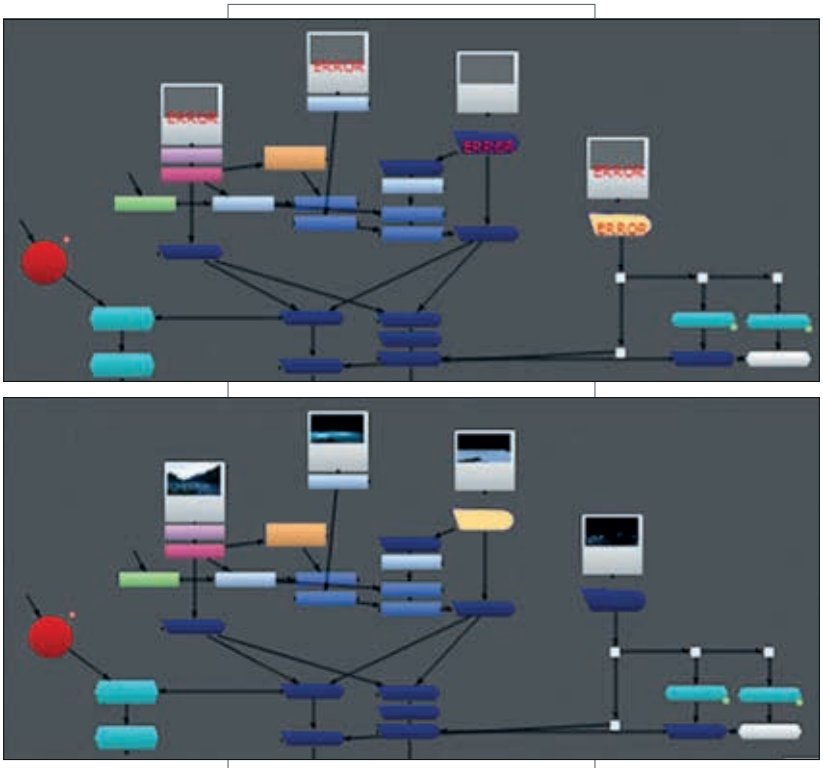


Abb. 04: Speicherpfad in allen Nodes umstellen

Code Snippets

Nachdem Sie nun bewaffnet sind mit dem Wissen der anderen, Ihre Projektkonventionen aufgestellt haben und die Templates bereit liegen, ist es an der Zeit, kleine Skripte in den Alltag zu integrieren. Dafür sind keine tieferen Programmierkenntnisse nötig. Wichtig ist nur, dass Sie folgende Punkte verstehe:

a) Was macht das Skript?
b) Wie und wo kann ich es anpassen?

Für die folgenden Beispiele nutzen wir vor allem die Programmiersprache Python, da sie in den meisten Paketen wie Maya, Nuke, Houdini, Cinema 4D usw. integriert ist. Um die Skripte zu benutzen, öffnet man den jeweiligen Skripteditor in der jeweiligen Software, wählt Python als Sprache aus und drückt ausführen (Enter oder Strg + Enter, je nach Software). Sollte Sie eines der Beispiele ansprechen, probieren Sie es direkt aus. Um die Befehle sinnvoll bereitzustellen, sollten Sie sie in einer Datei speichern, einen Shelf-Button daraus kreieren (Maya) oder sie in Menu bzw. Toolbar einbetten.

General

Eine der wichtigsten Aufgaben bei Pipelinesystemen sind Pfade. Das Navigieren zu bestimmten Orten im Projekt ist einer der größten Zeitfresser im Alltag. Dabei gibt es Pfade, die man immer wieder öffnet, wie Review, Animationplayblast, Render usw.

Öffnen von festen Ordnern und Webseiten

Die Standardbibliothek Webbrowser ist hauptsächlich zum Öffnen von Weblinks gedacht, jedoch können wir sie auch zum Öffnen von Pfaden nutzen, was eine Doppelfunktion erlaubt – siehe **Abbildung 01**.

a) Öffnen von Links im Standardbrowser und Pfaden im Explorer.
b) Bei „path" den Inhalt der Anführungszeichen ersetzen. (Nur ein Path ist notwendig, der andere ist nur zum Vergleich zwischen Link und Pfad da.)

Öffnen des aktuellen Szenenpfads

Nachdem wir gesehen haben, dass „webbrowser" feste Pfade öffnen kann, wäre es weitaus nützlicher, den aktuellen Szenenpfad im Explorer öffnen zu können. Dafür müssen wir die jeweilige Bibliothek der Software importieren und dann die path-Variable mit dem aktuellen Szenenpfad füttern – zu sehen in **Abbildung 02**.

a) Öffnen des aktuellen Szenenpfads im Explorer.
b) Nur den jeweiligen Softwarecode übernehmen. (Bei Nuke müsste der Maya- Code [eingerahmt in *] entfernt werden, sonst gibt es einen Fehler.)

Erstellen von Screenshots

Softwarepakete wie Maya, Nuke und Houdini bedienen sich an der Grafikoberflächenbibliothek PySide. Diese erlaubt, neben der Erstellung grafischer Oberflächen auch Screenshots vom Hauptmonitor zu machen und sie an einem vorgegebenen Ort zu speichern (siehe **Abb. 03**).

a) Erstellen von Screenshots vom Hauptmonitor
b) Speicherpfad in „path" ersetzen (Bilddatei wird überschrieben)

```python
# Batch: Change read path
node_type     = 'Read'
old_path_part = r'I:'
new_path_part = r'D:\project'

for read in nuke.allNodes(node_type):
    path = read.knob('file').getValue()
    path = path.replace(old_path_part, new_path_part)
    read.knob('file').setValue(path)
```

Abb. 05: Ersetzen alter durch neue Pfadteile

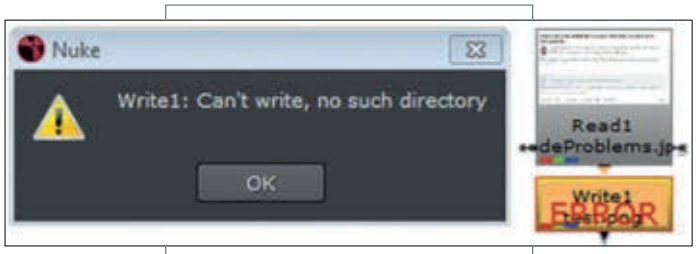


Abb. 06: „Error" beim Rendern eines Write-Nodes

Nuke

Nuke selbst besitzt eine starke Python-Anbindung und erlaubt, ganze Node-Bäume damit zu erstellen. Wir konzentrieren uns jedoch auf die täglichen Herausforderungen.

Abb. 07: Write-Node im Python-Tab

Lade- oder Speicherpfad in allen Nodes umstellen

Ein typisches Problem beim Öffnen fremder Szenen oder an einem anderen Ort: Die absoluten Pfade der Read-Nodes stimmen nicht mehr. Nun gilt es, den Unterschied zu finden und jede einzelne händisch zu ersetzen. Das geht auch einfacher. Sind die Bilder gleich strukturiert, aber der Ordner an einer anderen Stelle, ist es möglich, in allen Nodes den alten Teil durch den neuen zu ersetzen (Find & Replace) – zu sehen in **Abbildung 08**.

a) In allen Read-Nodes alte Pfadteile durch neue ersetzen.

b) In old_path_part den alten Pfadteil ändern, während in new_path_part der neue einfügt wird.

Wer bei „node_type = ‚Write'" eingibt, ändert beim Ausführen den Pfad aller Write-Nodes. Sollten Sie Backslashes bei Pfaden nutzen, z.B. „D:\project", müssen Sie ein „r" (raw) davor setzen: „r'D:\project'" (Python erwartet nach einem Backslash ein Sonderzeichen und würde den Pfad zerstören.)

```python
# CREATE folder [before render]
if not os.path.isdir(os.path.dirname(nuke.thisNode()['file'].evaluate())):
    os.makedirs(os.path.dirname(nuke.thisNode()['file'].evaluate()))
```

Abb. 08: Codezeilen für die Erstellung der Speicherpfadordner (before render)

```python
# CREATE folder prewrite
import nuke, os, errno

def create_write_dir():
    file_name = nuke.filename(nuke.thisNode())
    file_path = os.path.dirname(file_name)
    os_path   = nuke.callbacks.filenameFilter(file_path)

    try:
        os.makedirs(os_path)
    except OSError, e:
        if e.errno != errno.EEXIST:
            raise

nuke.addBeforeRender(create_write_dir)
```

Abb. 09: Ordnererstellung in Write-Nodes integrieren

Erstellen vom Renderordnern

Der Versuch, einen Write-Node zu rendern, ohne den dazugehörigen Speicherort zu haben, endet in einer Fehlermeldung. Wieso The Foundry dafür nicht zumindest eine Option anbietet, ist verwunderlich, aber nichtsdestotrotz lösbar.

Der schnellste Weg wäre, im Python-Tab des Write-Nodes, unter „before render", die zwei Zeilen Code einzufügen. Damit wird für diesen Write-Node sichergestellt, dass das Verzeichnis erstellt wird, bevor das Ergebnis gerendert wird. Der wesentlich effektivere Weg wäre, diese Änderung für alle Write-Nodes als Standard zu definieren. Genau das geschieht durch den Befehl „nuke.addBeforeRender(create_write_dir)".

```python
# RENDER viewer snapshot
def nuke_viewer_snapshot(path):
    import nuke
    viewer   = nuke.activeViewer()
    actInput = nuke.ViewerWindow.activeInput(viewer)
    selInput = nuke.Node.input(viewer.node(), actInput)

    if actInput < 0: return False

    # create writes & define render format
    write1 = nuke.nodes.Write(file=path.replace("\\","/"), name='writeNode1', file_type='jpg')
    write1.setInput(0, selInput)

    # look up current frame
    currentFrame = int(nuke.knob("frame"))
    # render
    nuke.execute( write1.name(), currentFrame, currentFrame)
    # clean up
    for n in [write1]:
        nuke.delete(n)

nuke_viewer_snapshot(r"C:\snapshot_nuke.jpg")
```

Abb. 10: Viewer-Snapshot

```
# Disable Viewport (for export)
import maya.mel
# viewport OFF
mel.eval("paneLayout -e -manage false $gMainPane")
# viewport ON
mel.eval("paneLayout -e -manage true $gMainPane")
```

Abb. 11: Maya Viewport aktivieren/deaktivieren

```
def maya_viewport_snapshot(path):
    import maya.cmds, maya.mel
    mel.eval('setAttr "defaultRenderGlobals.imageFormat" 8;')

    # playblast one frame to a specific file
    currentFrame = str(cmds.currentTime(q=1))
    snapshotStr = 'playblast -frame {} -format "image" -cf "{}" -v 0 -wh 1024 576 -p 100;'.format(currentFrame, path)
    mel.eval(snapshotStr)

    # restore the old format
    mel.eval('setAttr "defaultRenderGlobals.imageFormat" `getAttr "defaultRenderGlobals.imageFormat"`;')

maya_viewport_snapshot(r"C:\snapshot_maya.jpg")
```

Abb. 12: Maya Snapshot

Um diesen Befehl nicht jedes Mal neu einzugeben – Nuke ist vergesslich – macht es Sinn, ihn in die „init.py" zu schreiben, damit er bei jedem Start ausgeführt wird – siehe dazu **Abb. 09**.

a) Erstellen von Renderpfaden für den Write-Node.

b) Die einfache Version: in „before render" des jeweiligen Write-Nodes einfügen, oder die komplexere Version: bei jedem Neustart ausführen bzw. in der „init.py" ausführen lassen.

Aktuellen Viewer-Snapshot erstellen

Immer wieder gibt es Situationen, in denen es praktisch wäre, die aktuelle Anzeige als Snapshot zu haben, um sie sich nochmal in Ruhe anzuschauen oder zu teilen. Im Normalfall müsste man an dieser Stelle einen Write-Node erstellen, den Pfad zuweisen und rendern. Das folgende Skript übernimmt all diese Aufgaben und löscht nach dem Rendern den hinzugefügten Write-Node wieder – siehe **Abb. 10**.

a) Rendert den aktuellen View.

b) Anpassung des Pfads in der letzten Zeile. (Bildformatänderungen sollten auch in der Variable „file_type" angepasst werden.)

Maya – Viewport deaktivieren

Bei Prozessen wie dem Exportieren von Alembic, in dem sich die Timeline bewegt, wird nach jeder Frameänderung auch der Viewport aktualisiert. Es gibt eine Möglichkeit, diese Aktualisierung für den Moment auszuschalten und je nach Länge des Exports bis zu 20% an Zeit zu sparen – siehe **Abb. 11**.

a) Temporäre Viewportdeaktivierung für Prozessbeschleunigung.

b) Vor dem Prozess/Exportieren den Import-Befehl mit dem 1. Befehl ausführen, damit sich der Viewport nicht mehr automatisch aktualisiert. Nach der Fertigstellung, den 2. Befehl ausführen, um die Aktualisierung wieder zu aktivieren. (Das #-Symbol vor dem Befehl lässt diesen vom Programm unangetastet: „# mel.eval".)

Erstellen eines Viewport-Snapshots

Ähnlich wie beim Nuke-Snapshot ist es manchmal praktisch, den aktuellen Viewport als Bilddatei zu speichern – siehe **Abb. 12**.

a) Den aktuell selektierten Viewport rausrendern.

b) Den Speicherpfad in der untersten Zeile ersetzen. (Es ist auch möglich, die Auflösung (siehe –wh) und die Qualität (siehe –p) anzupassen. Sollte das Speichern nicht funktionieren, versuchen Sie es mit Forwardslashes: „C:/project".)

Ich hoffe, dass Ihnen die Ideen und Codeabschnitte helfen werden, schneller in der Umsetzung Ihrer Visionen zu sein, oder Sie beflügeln, im Netz nach weiteren Hilfen zu suchen. **› ei**

Alexander Richter ist Technical Director für Look & Software Development, Lighting und Pipeline. Er studiert am Animationsinstitut Technical Directing und arbeitet als Lead Pipeline TD bei Studio Soi in Ludwigsburg und als Dozent für Animation an der media Akademie (mAHS) in Stuttgart.

www.richteralexander.com

Links

Code Snippets
▷ richteralexander.com/src/pfa.py

Pipeline
▷ github.com/richteralexander/plex/wiki

Pipelineartikel
▷ goo.gl/CzjUCX