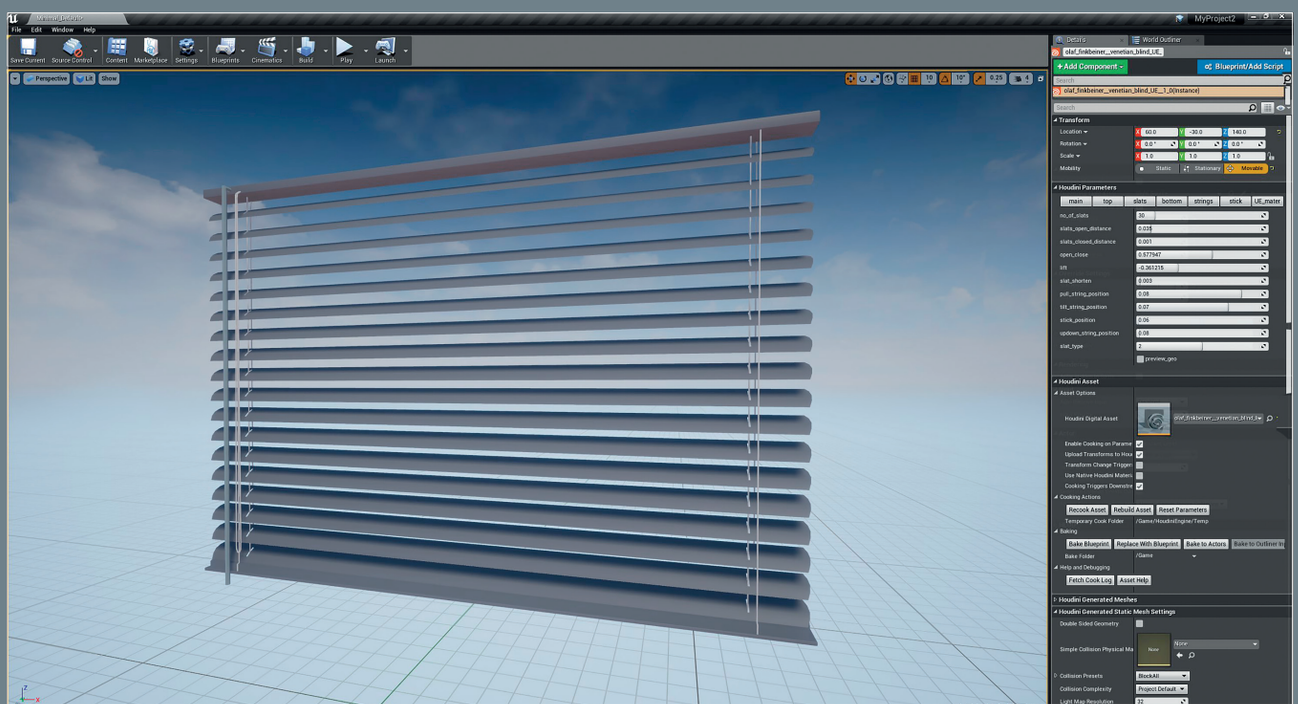


Houdinis Asset-Zauberkasten –

Houdini Digital Assets und die Houdini Engine in UnrealEngine

Assets im klassischen Sinne sind Texturen, Geometrien und Szenen – also .jpg- und Photoshop-Dateien oder Pinsel, aber auch HDR-Sphären und Render Output. Ich würde soweit gehen und Vorlagen, Texte, Konzepte und Referenzmaterial dazu zählen. Das alles liegt in der Regel verteilt auf File-Servern, lokalen Festplatten, externen Datenträgern und gerne auch mal in der Cloud. Alle Daten natürlich häufig auch in etlichen Versionen und Reifegraden. von Olaf Finkbeiner

Powered by SAE INSTITUTE

Um das gleich vorweg zu sagen: Houdini ist keine Asset-Management-Software. Houdini ist vieles, aber das nicht. Nichtsdestotrotz sind Houdini Digital Assets wertvolle Assets – und können sich auch durch die Pipeline bewegen. Beispielhaft werde ich Ihnen zeigen, wie dank der Houdini Engine ein HDA (Houdini Digital Asset) in der Unreal Engine eingesetzt werden kann. Dazu verwende ich als Beispiel-HDA eine Jalousie (auf englisch „Venetian Blind"). Die Idee, etwas so wiederverwendbares als HDA zu erstellen, entstand im Gespräch mit einem Archiviz-Freund. Übrigens: Alle Bilder zu diesem SAE-Workshop können Sie in voller Auflösung unter **www.digitalproduction.com** herunterladen, um jedes Detail zu sehen.

Asset Management in Houdini

Houdini zwingt einem geradewegs auf, die in einem Projekt verwendeten Assets zu managen. Und zwar ermöglicht es, Assets in die Szene einzubinden, ohne diese mit der Szene abzuspeichern. Die Assets liegen in Ordnern und werden mit dem File Node geladen. So bleiben Houdini-Szenen leicht und speichern schnell. Das gilt auch für bereits bestehende Assets, die als .fbx, Alembic oder .bgeo, dem Houdini eigenen Format, gespeichert sein können.

Werden die Geometrien erst durch Houdini erzeugt, gibt es den äußerst praktischen File Cache Node. Dieser ermöglicht es, eine Geometrie oder auch ein Volumen oder Partikel aus der Szene heraus in einem Ordner zu speichern. Beim nächsten Öffnen der Szene muss die Geometrie jetzt nur noch geladen und nicht erneut berechnet werden. Das macht immer dann Sinn, wenn das Generieren der Geometrie viel Zeit in Anspruch nimmt und somit länger dauert, als zu laden. Dabei können auch Animationen Frame für Frame auf der Platte verewigt werden.

Verpackte Geometrie

Houdini hört da aber nicht auf, das Konzept der „Packed Geometrie" setzt noch einen oben drauf. In einer „Packed Geometrie" werden die Daten in einen Container gepackt. Dieser ist dann nur noch ein Point, der mit frei zu definierenden Attributen angereichert ist. Dies können Ausrichtung, die Größe

der Bounding Box oder auch eine Gruppenbezeichnung sein – oder welche Metadaten auch immer. Dieses Konzept ermöglicht es, Tausende oder Millionen von Objekten zu platzieren, ohne dass dafür Unmengen von Daten berechnet werden müssen. Dabei sieht man im Viewport entweder nur einen Punkt, eine Bounding Box, eine Point Cloud oder auch die volle Geometrie, je nachdem was die Grafikkarte leisten kann. Das geschieht jedoch nicht automatisch, sondern muss vom Artist entschieden werden. Beim Rendern wird selbstverständlich die volle Geometrie gezeigt.

Folder Structures und lokale Variablen

Eine Houdini-Szene besteht also oft aus der Szene (.hip) und den dazugehörigen Assets. Die Assets liegen in Ordnern – und wo diese Ordner sich befinden, wird in einer frei zu definierenden Local-Variable gespeichert. So kann zum Beispiel die Variable „$textures" eben den Pfad zu den Texturen enthalten. Auf die konkrete Textur würde dann mit „$textures/wood/oak1/diffuse.png" im Shader zugegriffen – und auch Ausgabepfade oder Namenskonventionen können so für alle im Team gemanagt werden. Dazu braucht es dann vielleicht schon kleine Python-Skripte. Diese zu erstellen ist aber ein Kinderspiel, da das grundsätzliche Houdini-Konzept dies schon vorweg denkt. Mehr dazu hier: **bit.ly/jurjuraj_tomori_environment_variables**.

Batch

Assets werden auch oft in verschiedenen Versionen benötigt. Das können LODs sein oder Kollisionshüllen für Ridgid-Body-Simulationen oder Game Engines oder Vorschaubildchen etc. – und auch hier kann Houdini prima bei der Batch-Konvertierung eingesetzt werden.

Houdini Digital Asset

HDAs (Houdini Digital Assets) sind die nächste Evolutionsstufe. Houdini ist selbst zu großen Teilen aus HDAs zusammengesetzt, diese sind häufig sogar Open Source und können somit für eigene Zwecke angepasst werden. HDAs innerhalb von Houdini und Mantra können aber vieles sein: Ein HDA kann eine Geometrie ausgeben oder eine Geometrie bearbeiten (dabei nicht zu vergessen: In Houdini sind eben auch Volumes und Partikel eine Geometrie).

Ein HDA kann aber auch Assets beinhalten, also Geometrie oder Texturen einbinden und durch Skripte (Python) beliebig erweitert werden. Beim Erstellen kann auf viele vorgefertigte sogenannte Handles zugegriffen werden. Handles – manchmal auch Gizmos genannt – können genutzt werden, um im Viewport etwas einzustellen. Dazu später mehr im Teil über UnrealEngine/Houdini Engine.

Doch auch das ist nicht alles: HDAs können auch als „Engine Procedurals" in Mantra eingesetzt werden. Dann wird die Geometrie erst beim Rendervorgang erzeugt. Wie das funktioniert, lesen Sie dann im Mantra-Artikel im Renderer-Schwerpunkt der nächsten DP-Ausgabe.

Versioning, Namespaces & Asset Management

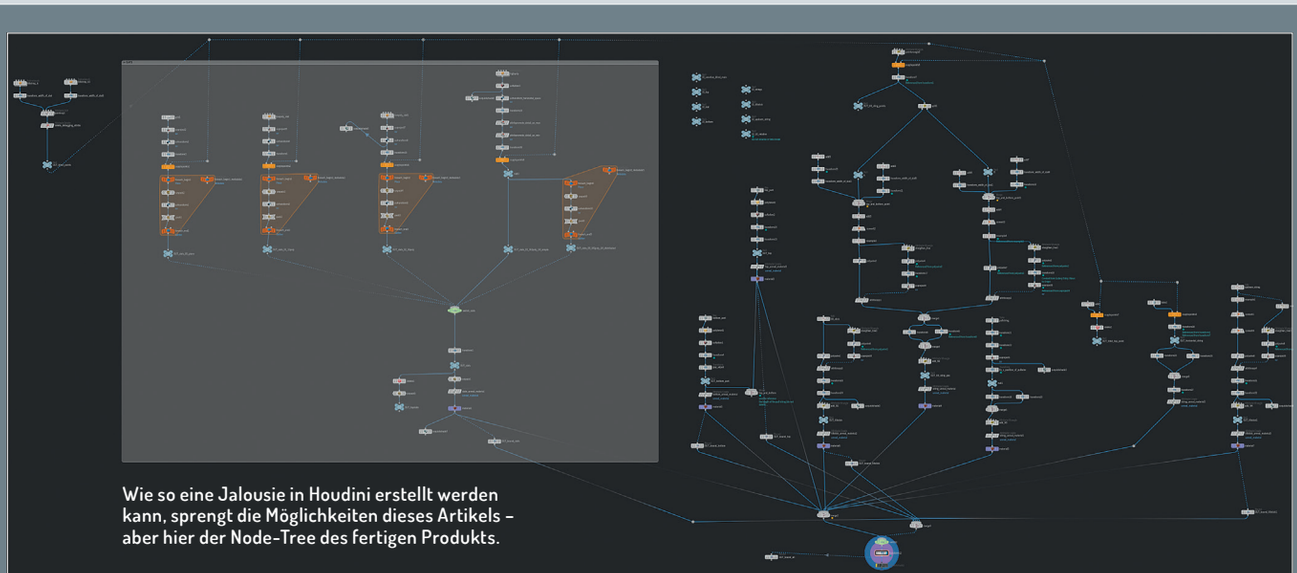
Versioning von HDAs ist auch vorgesehen. Dazu gibt es grundsätzlich zwei Vorgehensweisen. Wer es genau wissen will, einfach in der Houdini Help nach „Two types of asset versioning" suchen. Ich nutze die Variante, bei der im Operator-Namen die Version gespeichert wird. Die Regel dabei ist: [namespace::]node_name[::version]. Beispiel: „com.digitalproduction::venetian_blind::1.0".

Der Namespace ist wichtig, um eigene Assets, die ja zufällig genauso heißen könnten, von denen anderer Entwickler und Artists zu unterscheiden.

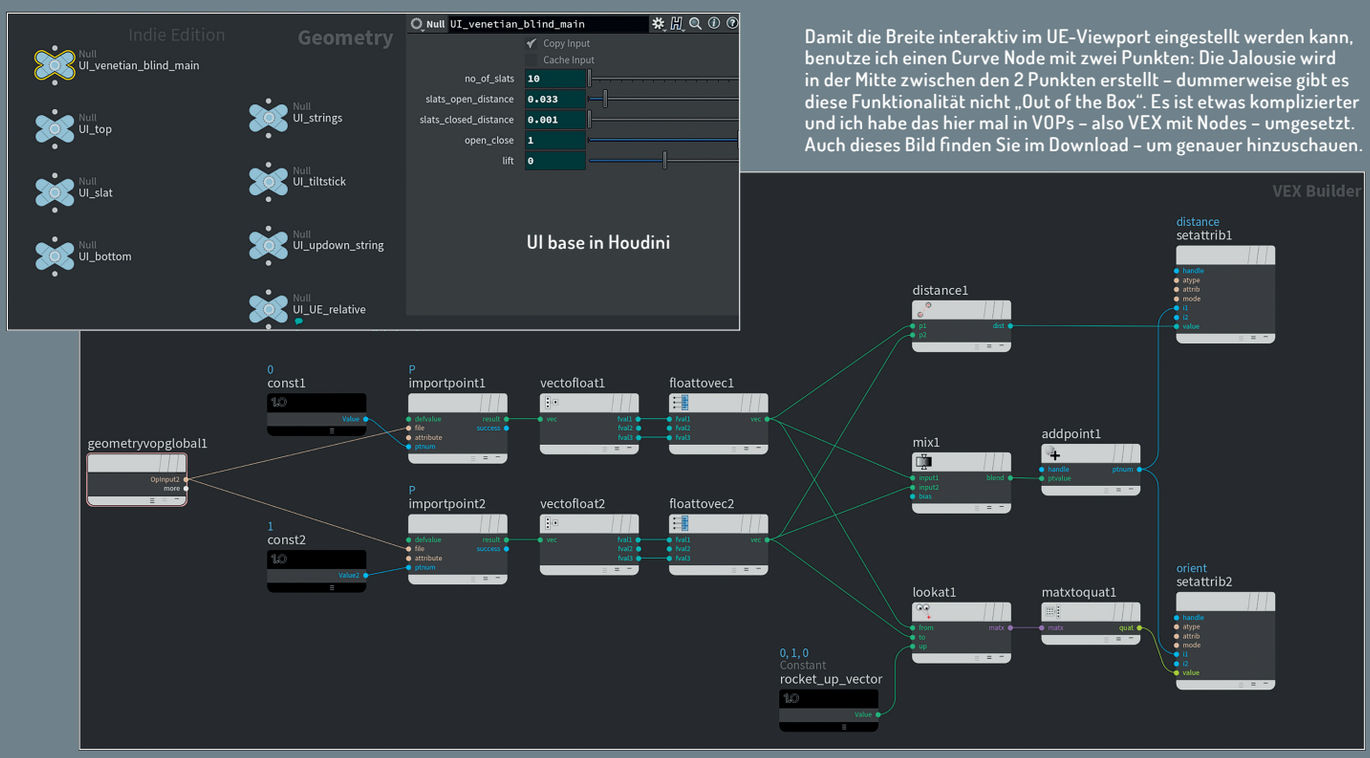
Unreal Engine, Houdini Engine und HDAs dazwischen

Um jetzt so ein HDA in der Unreal Engine zu verwenden, benötigt man eine Houdini-Engine-Lizenz und die Houdini Engine muss separat heruntergeladen werden. Die Houdini Engine ist sozusagen ein Houdini ohne User Interface, dafür aber mit API – einer Schnittstelle zur Anwendungsprogrammierung. Als Houdini-Indie-User bekommt man bis zu 3 Engine-Lizenzen gratis dazu, bzw. bis zu 3 Houdini-Engine-Lizenzen bekommt jeder gratis auch ganz ohne Houdini-Lizenz. Abgesehen von der Schnittstelle zu Unreal, Maya, Unity und C4D ist die Engine auch in der Lage, mit Mantra zu rendern oder Fluids und Pyro zu simulieren. In einem größeren Studio muss also nicht für jeden Rechner, der Wassersimulation berechnen soll, eine Vollversion von Houdini installiert sein – es reicht eine deutlich günstigere Houdini-Engine-Lizenz, die im Batch Mode läuft. Der Preisunterschied: 500 US-Dollar pro Jahr (Rental) im Gegensatz zu 4.500 US-Dollar (Perpetual).

Zurück zu HDAs in UE. Die meist gezeigten Beispiele sind: Treppen, Brücken, Zäune, Pflanzen etc. Ich habe hier als Beispiel eine Jalousie ausgewählt – ich bin mir sicher, wer viel Architekturvisualisierung macht, kann bestätigen, dass es dafür Bedarf gibt. Zudem ist so eine Jalousie zwar ein Alltagsgegenstand, bei näherer Betrachtung aber eine



Wie so eine Jalousie in Houdini erstellt werden kann, sprengt die Möglichkeiten dieses Artikels – aber hier der Node-Tree des fertigen Produkts.



Damit die Breite interaktiv im UE-Viewport eingestellt werden kann, benutze ich einen Curve Node mit zwei Punkten: Die Jalousie wird in der Mitte zwischen den 2 Punkten erstellt – dummerweise gibt es diese Funktionalität nicht „Out of the Box". Es ist etwas komplizierter und ich habe das hier mal in VOPs – also VEX mit Nodes – umgesetzt. Auch dieses Bild finden Sie im Download – um genauer hinzuschauen.

recht komplexe Angelegenheit. Sie besteht aus den Lamellen, verschiedenen Schnüren, einem Kasten oben, einem Stab zum Einstellen der Neigung und einem Seil für die Höheneinstellung. Jalousien werden zudem in verschiedenen Breiten und Längen benötigt. Und damit sie im Rendering auch schön aussehen, will man sie hoch- und runterlassen und die Neigung der Lamellen einstellen können.

Tutorials!

Dieses „Venetian Blind"-HDA erzeugt ein Geometrieobjekt – aber HDAs können in Unreal auch bestehende Elemente nutzen und bearbeiten. Ein wirklich gutes Video dazu ist **bit.ly/HD_Andreas_Glad** „EUE 2017: Andreas Glad – Assets That Care: Why I'm Rebuilding all my Realtime VFX Workflows in Houdini". Hierauf gehe ich in diesem Artikel nicht wirklich ein und deshalb sollte es unbedingt angesehen werden.

Apropos Video-Tutorials: Varomix hat kürzlich ein sehr gutes Tutorial veröffentlicht. Es zeigt den kompletten Entstehungsprozess eines HDAs für Unreal und ist eher für Houdini-Anfänger. Sein pragmatischer Stil ist sympathisch, und dem Metal-Fan aus Mexiko zuzuhören ist ein wenig wie Cheech und Chong zuzuhören. Er vermeidet VEX und Coding soweit es geht, macht Fehler und verbessert diese dann gekonnt – am besten zuerst eines seiner kostenfreien Videos ansehen: **http://bit.ly/varomix_hda_ue**.

HDA für Unreal vorbereiten

Nun geht es los: Ich bevorzuge es, ohne HDA-Funktionalität das jeweilige Tool zu entwickeln und dieses modular zu gestalten. Jedes Bauteil des Assets bekommt ein eigenes User Interface, wobei ich jeweils einen Null Node erstelle, der als Träger für das UI dient. Die Parameter sind, soweit es geht, auch relativ angelegt, das heißt sie beziehen sich aufeinander. Der Hauptparameter ist in unserem Beispiel die Breite der Jalousie. Andere Parameter wie die Breite der Lamellen oder der Abstand der Seile sind dann relativ zur Gesamtbreite. So muss der Anwender dann nicht etliche Parameter einzeln justieren, wenn die Breite der Jalousie an das Fenster angepasst wird.

Was passiert in der VOP?

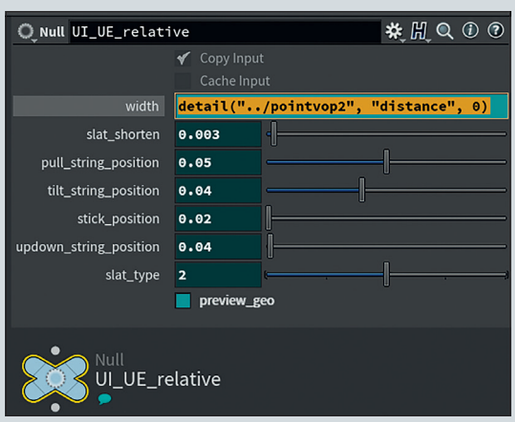
Der VOP Node läuft dazu nicht über jeden Point sondern nur einmal (Detail(once only)). Von OpInput2 werden 2 Point (P)ositions importiert. Von diesen wird Y nicht genutzt (die Höhe Y ist hier nicht wichtig), deswegen vectofloat und floattovec.

Mit dem Distance Node bekomme ich den Wert für die Breite der Jalousie, mit dem Mix Node bekomme ich die Mitte zwischen den zwei Punkten (linear interpolation). Und an der Stelle erzeuge ich einen neuen Point. Mit dem Lookat Node bekomme ich eine Matrix der Drehung, diese wandle ich in ein Quaternion um, um den Wert dann als @orient-Point-Attribut auf den erzeugten Mittelpunkt zu speichern.

Die Distance speichere ich als Detail-Attribut, welches vom UI für die Breite ausgelesen wird – Hscript code: detail(„../pointvop2", „distance", 0).

Der Copytopoints Node benutzt den einen gerade erzeugten Punkt, um darauf bzw. mehr darunter, da der Pivot oben in der Mitte der Jalousie sein soll, eine Jalousie zu platzieren. Dabei wird automatisch das @orient-Attribut ausgelesen und die Jalousie passend gedreht.

Tipp: @orient ist wirklich ein essentielles Attribut, wenn man in Houdini prozedural modellieren oder Partikel auf die Reise schicken will. Wer VEX lernen will, hier der beste Link: „The Joy of Vex" von Matt Estela unter **bit.ly/JoyofVex**.



1,2,3 ... HDA für UE erstellen

Jetzt wo das Asset bzw. Tool in Houdini fertig ist, muss es als HDA verpackt werden, um in UE via Houdini Engine eingesetzt werden zu können – dies ist aber überraschend einfach. Im Menü „Assets" wählen wir „New Digital Asset From Selection ...", legen Name, Label und Speicherort fest und klicken auf „OK". Sollte hier die abgebildete oder eine ähnlich lautende Fehlermeldung auftauchen, macht man am besten mit dem Snipping Tool einen Screenshot und speichert trotzdem – der Fehler kann später ausgebügelt werden. Und fertig ist das HDA! Das war der einfache Teil ...

User Interface für unser HDA in Unreal

Nach dem Speichern hat das HDA jedoch kein User Interface, denn das eigentliche UI des HDAs wird in den Type Properties erstellt und definiert, was mit dem HDA gemacht werden kann.

Wichtig: Nicht mit dem fast identisch aussehenden Menü mit dem Zahnrad(Gear)-Icon verwechseln! Die einzelnen Parameter können per Drag-and-drop von den vorher angelegten UI Null Nodes in das HDA-Interface übernommen werden, die aktuell eingestellten Werte und Parameter-Ranges werden im HDA als Preset übernommen. Dort können sie zudem in Ordnern organisiert werden, die dann auch in UE angezeigt werden. Oder sie werden auf dem „from nodes" in das UI des HDA übernommen.

Handles & Editable Nodes

Für die Jalousie habe ich mich dazu entschieden, die Platzierung in UE durch ein editierbares Curve-Objekt zu implementieren. Die Kurve hat dabei nur zwei Punkte, zwischen diesen Punkten wird die Jalousie eingepasst, sie wird also automatisch genau so breit wie der Abstand zwischen den zwei Endpunkten der Kurve. Dazu muss in „Edit Operator Type Properties" der Node in „Editable Nodes" eingetragen werden. Auf die gleiche Art und Weise würden auch HDAs zum Beispiel für Zäune, Kabel oder anderes, was an einer Kurve oder Linie ausgerichtet werden soll, in UE eingebunden werden.

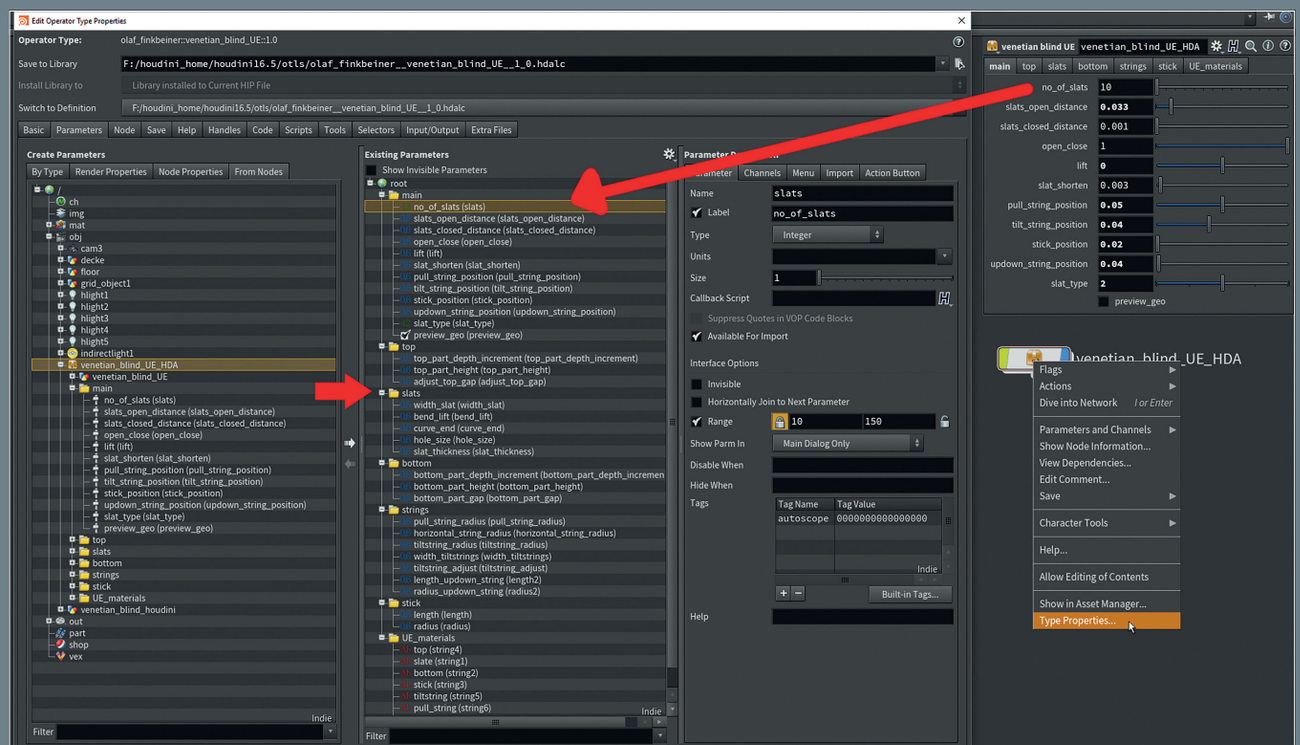
Eine weitere Möglichkeit stellen Handles dar. In Houdini kann ein HDA sehr viele verschiedene Handles haben, für UE stehen momentan nur zwei zur Verfügung. Das sind „Bounder" und „Transform", diese können dann aber auch mehrfach verwendet werden.

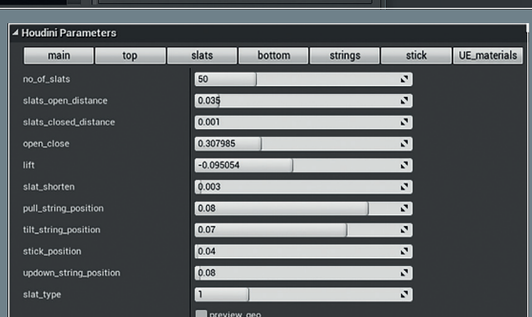
Um in UE das Handle benutzen zu können, muss man das Objekt auswählen, dann die Kiste durch Klicken auf die Kiste anwählen – dies wird dann rosa – und jetzt erst kann das Handle verwendet werden.

LODs Level of detail und Collider

In Games werden verschieden detaillierte Versionen von Geometrien benötigt, das Ganze nennt man LODs (Level of detail). Houdini erzeugt diese LODs nicht automatisch – Unreal kann das. Wer aber verschiedene LOD-Meshes erzeugt hat, kann diese an UE übergeben, indem er sie in Gruppen einsortiert. Die Gruppen müssen dann „lodX" (lod0, lod1...) benannt werden. Collider sind die Geometrien, die in UE als Hindernisse definiert werden. Unreal ermöglicht nur einen Collider, der dann für alle LODs gleichermaßen gilt. LOD-Einstellungen in UE können durch das Setzen von Detail Attributes in Houdini erstellt werden:

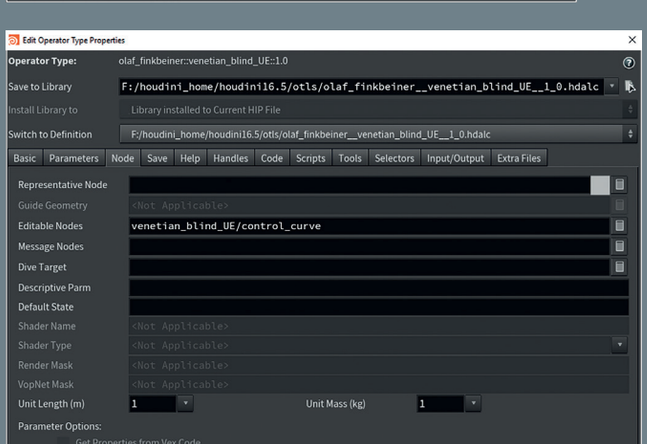
- Das LOD für den Collider durch das Detail Attribute: „unreal_uproperty_LODForCollision".
- Das Minimum-LOD durch das Detail Attribute: „unreal_uproperty_MinLOD".
- Die LOD-Gruppe durch das Detail String Attribute: „unreal_uproperty_LODGroup".

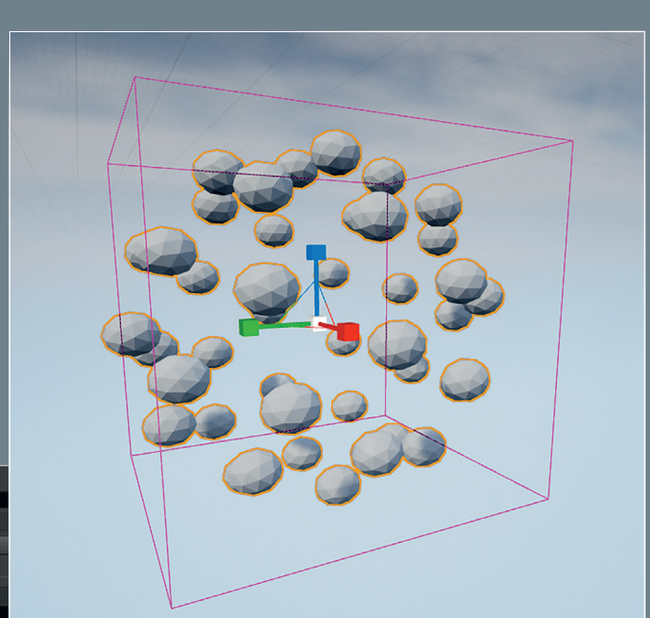


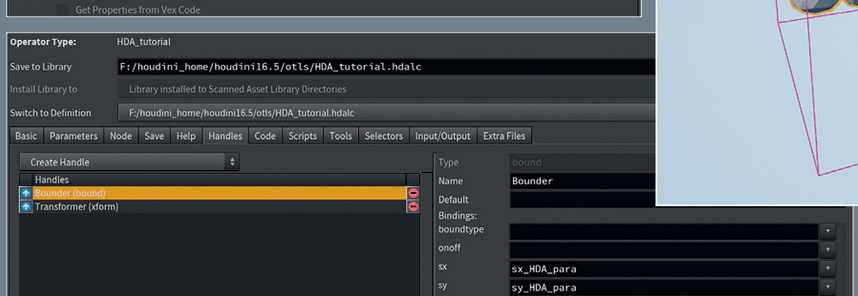


Ab in die Einstellungen: Hier finden Sie die Parameter für unser UI.

Das HDA inklusive Parametern in Houdini (links) und in Anwendung bei der UE (rechts)







▷ Screensize Values für jedes LOD können im Detail Attribute „lodX_screensize" gesetzt werden, oder durch ein „lod_screensize" Primitive Attribute. Bei manuell gesetzten Screensize Values wird „AutoComputeLODScreenSize" in UE abgeschaltet.

Für die „Ventian Blind" habe ich zusätzlich noch einen Preview-Knopf eingebaut, der dann nur Collision LOD ausgibt. Mit so einer einfachen Preview-Geo kann man dann die Objekte schneller platzieren, da die Berechnung dann schneller vonstattengeht.

Texturen

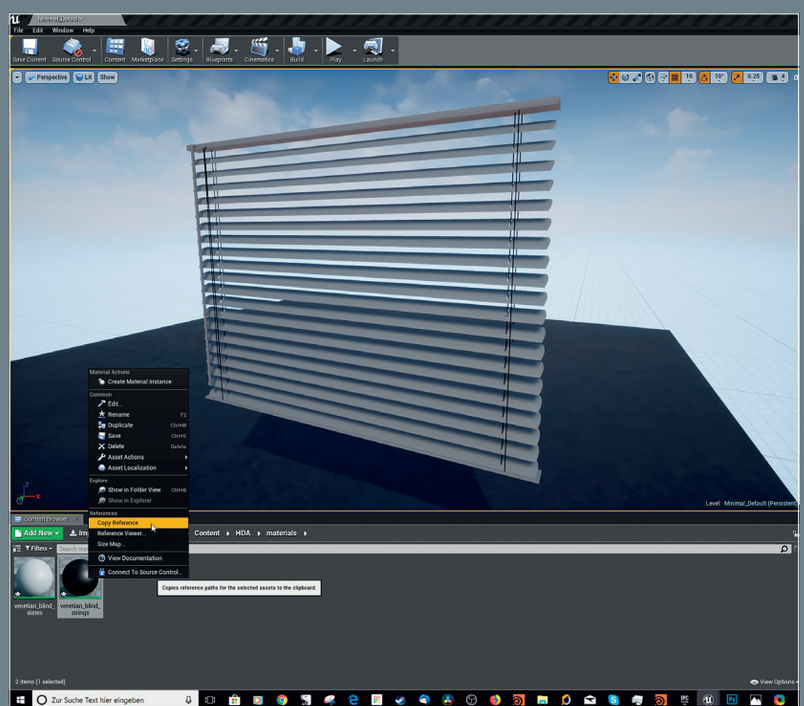
Texturen können in HDAs auch eingebunden werden. Diese müssen dann im Edit Operator „Type Properties" unter der Rubrik „Extra Files" ins HDA eingeladen werden. Das in Houdini erstellte Material nutzt diese dann. Noch viel cooler ist es aber, finde ich, dass Texturen mit prozeduralen Shadern auch von UE aus via Engine gebacken werden können. Dazu habe ich ein kleines Video-Quick-Tip-Tutorial aufgenommen (**bit.ly/houdini_quicktip**). Was dabei viele Houdini-Anfänger verwirrt, ist, dass innerhalb des HDAs verschiedene Kontexte hinzugefügt werden müssen. Es braucht ein Matnet (Materialnetzwerk), eine Kamera, unter Umständen ein Licht und eben eine ROP, also einen Render Operator, in dem dann der Bake Texture Node ist.

Wie geht's weiter?

Abschließend ist noch zu sagen, auch für Terrains wurde Houdini Engine nach Unreal-Engine-Schnittstelle implementiert. Dabei werden die Besonderheiten von UE Terrains berücksichtigt. Die Houdini-UE-Anbindung wird momentan sehr aktiv von SideFX entwickelt. Das Ganze scheint sehr gut anzukommen. Da SideFX dabei sehr transparent ist, einfach mal auf **sidefx.com/changelog/** gehen und unter Categories „Houdini for Unreal plug-in" auswählen. Neben aktuellen Neuigkeiten werden auch Bugfixes gelistet und man sieht, ab welchem Daily Build etwas verfügbar ist. Im Feb 2018 wurden zum Beispiel Funktionalitäten implementiert, die es ermöglichen, mit Houdini Packed Primitives und Unreal zu arbeiten – und diese erzeugen jetzt in Unreal eine hierarchische Struktur. Die oben beschriebenen LOD Groups funktionieren auch erst ab H16.5.372. **› ei**

Olaf Finkbeiner arbeitet als Senior Consultant im Automobilbereich. Er verfügt zudem über langjährige Erfahrung im Bereich 3D-Visualisierung. Nach Jahren der Praxis, hauptsächlich mit 3ds Max und VRED, hat er in Houdini und mit Mantra seine neue 3D-Software- und Renderer-Heimat gefunden.

Materialien



Es können Materialien in Houdini vergeben werden, diese werden dann in Unreal, so gut es eben geht, interpretiert. Ich finde aber die Möglichkeit, das Primitive Attribute „unreal_material" zu erstellen und dafür ein UI bereitzustellen, besser. Dort wird dann in UE der Materialpfad referenziert, dieser kann z.B. so aussehen: Material'/Game/HDA/materials/venetian_blind_slates.venetian_blind_slates'.