2021
2
ISSN 1433-2620 > 25. Jahrgang >> www.digitalproduction.com
Publiziert von Pixeltown GmbH

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

DIGITAL PRODUCTION

DIGITAL PRODUCTION

MAGAZIN FÜR DIGITALE MEDIENPRODUKTION

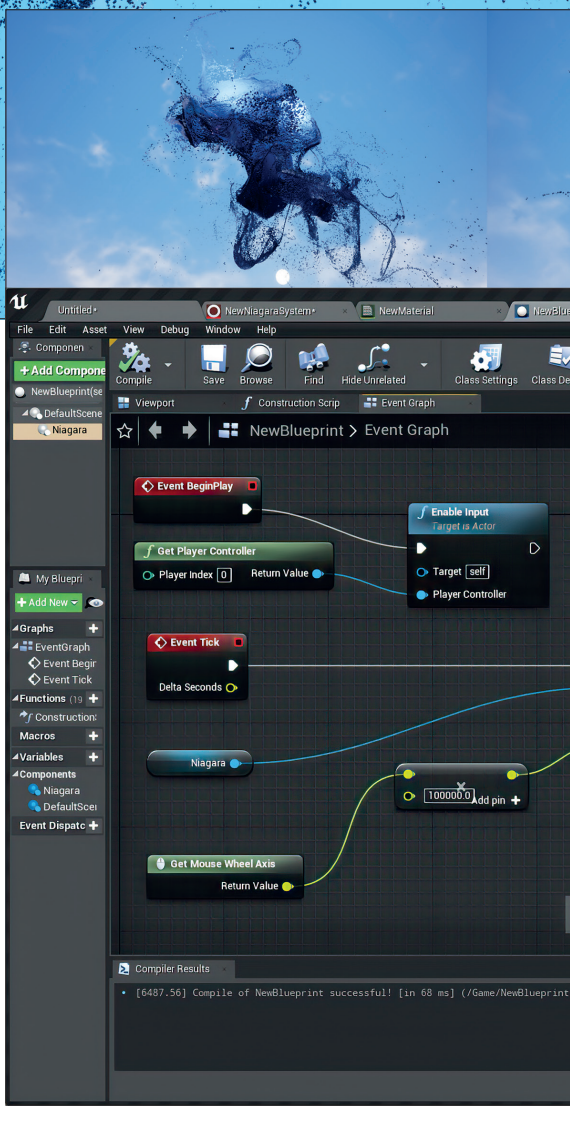
MÄRZ | APRIL 02:2021

Farbe
CineMatch, OmniScope und alles zu „bunt"

Vergnügen
Arbeiten von zu Hause, Retopo in 3ds Max oder Nuendo 11?

Hardware
Lenovo P620, Apple Macbook M1, Yolo Streaming

... und Software
Marmoset, Unreal Engine, Cascadeur, Rebelle und mehr



Laborunfall mit Unreal Niagara und Mesh Distance Fields

Mit Unreal Engine Release 4.26 ist das VFX-Partikelsystem Niagara (häufig gerne verkürzt als Niagra ausgesprochen) in vielen kritischen Punkten erheblich verbessert worden, nachdem es mit der Version 4.25 aus dem Early Access herauskam und seitdem das alte Cascade-System ersetzt. Insbesondere Designern eröffnet Niagara neue Möglichkeiten, die vorher nur via Coding umsetzbar waren.

von Cornel Hillmann

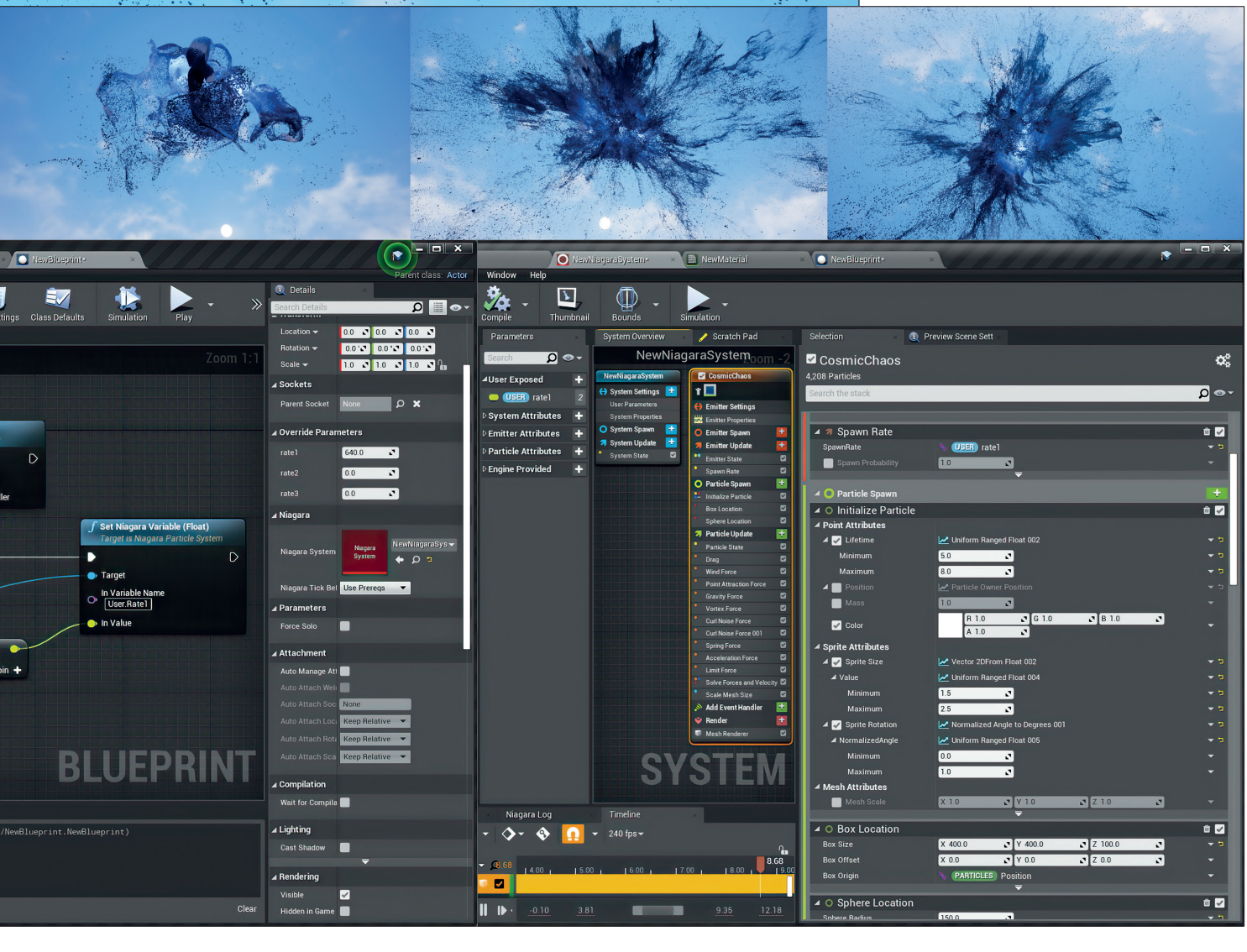
Das alte Cascade-System steht zwar weiterhin zur Verfügung (und kann sogar via Plug-in zu Niagara konvertiert werden), es ist jedoch in keinster Weise mit den Möglichkeiten von Niagara zu vergleichen. Zwar geht es im Kern um ähnliche Prinzipien, die man auch von anderen Partikelsystemen kennt, d.h. mithilfe von Emittern, Kräfteparametern und in der Regel hauptsächlich Sprites und dynamisch modifizierten Texturen partikelbasierte Effekte zu erzeugen. Jedoch greift Niagara wesentlich weiter als Cascade, indem es die Interaktion mit der Umgebung in fast jeder erdenklichen Weise via Visual-Scripting-Modulen steuern

Einige Hunderttausend Cubes als Niagara-GPU-Mesh-Partikel

lässt, und zwar in fast identischer Art, wie man es vom bewährten Blueprint-System her kennt.

Dass Niagara auf so viel Interesse in der VFX-Community stößt, hat natürlich einen Grund: Die neue Generation des Partikelsystems von Epic Games ist einer der Grundpfeiler für die Zukunft von Realtime VFX als Teil der angekündigten Unreal Engine 5 mit den neuen Technologien Nanite und Lumen, von der nicht nur Games- und Enterprise-Visualisierung, sondern insbesondere auch die Echtzeitgrafik in der Filmproduktion profitieren. Eine besondere Rolle spielt hier das neue Chaos Destruction bzw. Physics-System, das ursprünglich geplant PhyX als Default in 4.26 ersetzen sollte, jedoch dann im letzten Moment auf das nächste Release verschoben wurde. In engem Zusammenspiel mit der Chaosbruchbildung ist Niagara natürlich das entscheidende Bindeglied, um mit Rauch-, Staub- und Debris-Interaktionen die Level-Geometrie in Echtzeit zu verwüsten.

Steuerung der Partikel Spawn Rate in einem Blueprint Actor Niagara Component via Mausrad und User Variable „Rate1“. Diese wurde vorher im Niagara Editor als Custom Parameter angelegt und wird hier mit dem angelegten Maximalwert multipliziert und per Event Tick upgedatet.



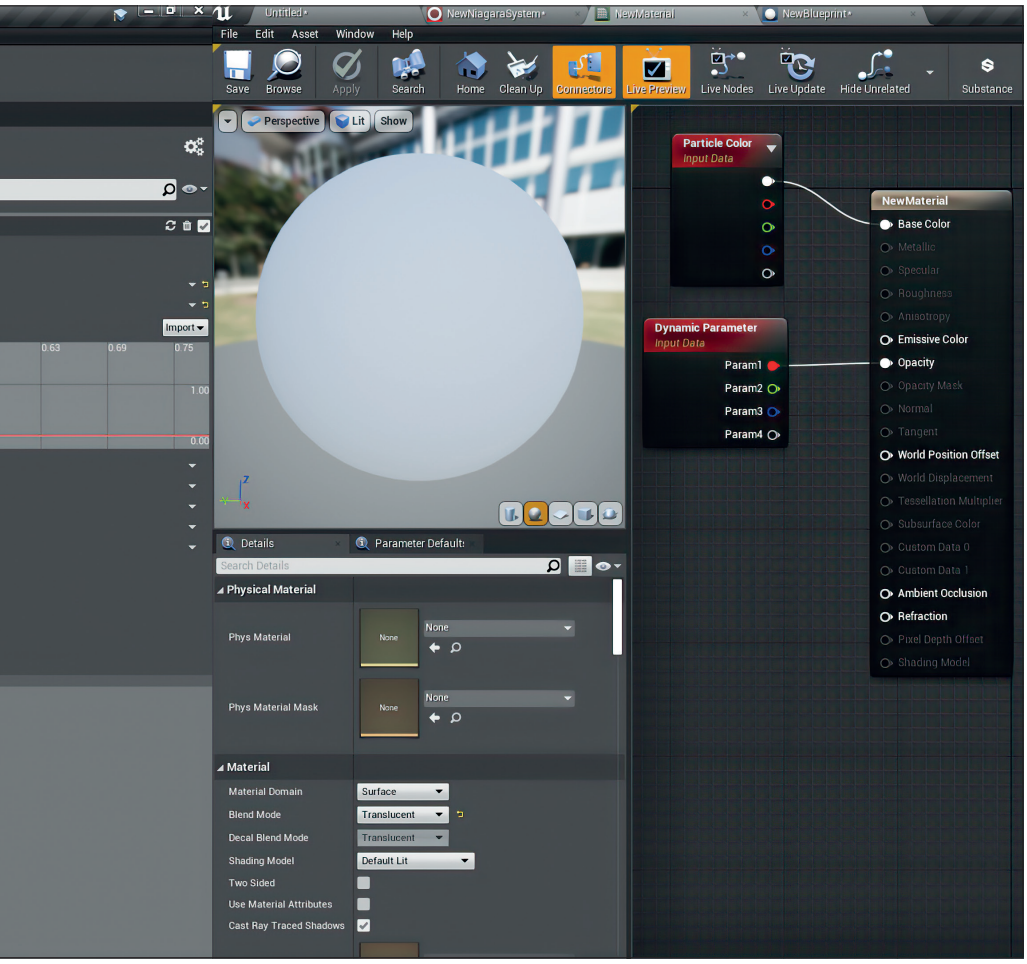
Dynamic Parameter im Material Editor (rechts) können direkt im Emittermodul abgegriffen werden, hier mit einer Kurve, die die Opacity der Partikel steuert (links).

Niagara auf der GPU

Die Revolution im Realtime-Filmmaking gibt also auch der Niagara-Entwicklung Rückenwind. Das wird besonders deutlich, wenn man einen Blick auf die wesentlich erweiterte und flexiblere GPU-Auslagerung von Partikeln wirft. Mit einer aktuellen Grafikkarte ist es somit problemlos möglich, mehrere Hunderttausend Mesh-Partikel zu zerwirbeln, ohne dass die Framerate aus dem grünen Bereich gerät (Anzeige via Konsole: stat fps). Zu der enormen GPU-Aufrüstung reihen sich dann noch die beinahe unbegrenzten Möglichkeiten, Partikelverhalten mit Szenendaten zu verknüpfen.

Am schnellsten lässt sich das testen, wenn man ein neues Niagara-System mit ein paar GPU-Emittern zusammenstellt, die Emitter mithilfe der neuen Turbulenzkräfte wie Curl Noise dynamisch verstreut und

Über „Map Get“ wird das Partikelalter mit dem Float Custom User Parameter „Power“ multipliziert und über den roten RGB-Kanal via „Map set“ upgedatet.



dieses System dann im Unreal Level via Blueprint Actor aufruft.

Sobald ein neuer Blueprint Actor erstellt, ein Niagara-System als Component hinzugefügt, Enable Input und Set Niagara Variable im Event Graph eingefügt sind, lassen sich nun Custom User Parameter, die in Niagara gesetzt wurden, im Blueprint Actor verknüpfen. Wenn z.B. die Particle Spawn Rate mit dem Mausrad in Echtzeit gesteuert werden soll, reichen ein Get Mouse Wheel Axis Node und ein Event Tick, da ja kontinuierliche Updates benötigt werden.

Das Mouse-Wheel-Beispiel zeigt, dass hier natürlich jegliche Art von Daten via User Parameter abgegriffen und als Niagara-Eingabe benutzt werden kann. Es ergibt sich also ein fast unendliches Potenzial, Positions-, Bewegungs- und Umgebungsdaten in die dynamische Partikelgestaltung einzubeziehen. Zusätzlich dazu lassen sich auch Materialattribute via Niagara steuern. Angefangen mit dem dynamischen Particle Color Node, der automatisch via Niagara Partikel-Color angesprochen wird, aber auch durch z.B. Dynamic Parameters, die in den Emittern aufgerufen werden können. Z.B. lässt sich die Opacity im Material Node so über eine Kurve im Niagara-Emitter steuern.

Scratch Pads zum Experimentieren

Man merkt, dass sich Epic Games viele Gedanken um die Benutzerfreundlichkeit des Systems gemacht hat. Übersichtliche Farbcodierung, Drag-and-drop, Copy-and-paste, automatische Fehlerkorrektur (via Tool Tip „Fix it"); es lässt sich zügig arbeiten, und beim Experimentieren kommt Freude auf. Ein Highlight sind dabei die sogenannten Scratch Pads. Mit ihnen lassen sich spontane Ideen unkompliziert testen. Ein Scratch Pad ist nichts anderes als ein temporäres Niagara Module Script, das man direkt im Emitter einfügen und testen kann. Nur wenn es sich bewährt, speichert man es als permanentes Asset im Projekt ab. Das Scratch Pad funktioniert genauso wie die vielen Module Scripts, die bereits in den Emittern als Library vorhanden sind. In diesen Skripten werden Partikelparametern via Map Get Node Eigenschaften und Verhalten zugeordnet und am Ende via Set Map Node in das System zurückgeführt. Was dazwischen via System und Custom User Parameter passiert, ist fast nur durch die Fantasie begrenzt. Am eindrucksvollsten demonstrierte dies vor einiger Zeit UE4 Evangelist Chris Murphy mit einer Character Disintegration Demo, die auf Oberflächenentfernungen der Umgebungsgeometrie via Mesh Distance Fields reagierte.

Schnell und unkompliziert lässt sich ein Scratch Pad oder Module Script testen, indem man es im Niagara-Emitter aufruft, einen Partikelparameter wie Particle Age mit einem User-Float-Wert multipliziert, um dann damit die Partikelfarbe über das Partikelalter zu modifizieren, indem der Ausgabewert in einen RGB-Farbkanal eingespeist wird. Was die neuen Möglichkeiten der Scratch Pads bzw. Module Scripts angeht, haben wir sicher bisher nur einen Bruchteil des Anwendungspotenzials gesehen. Es erwartet uns in Zukunft sicher so einiges an abstrakten Motion Graphics aus Niagara-Partikeln. Echtzeit-Visuals zum Beispiel, die auf Events und Konzerten via Niagara-Audio-Spectrum-Parametern auf Live Sound reagieren.

Nach der langen Beta-Phase ist Niagara inzwischen production-ready und auch flexibel für Datenschnittstellen. Man kann auf fast alles bis hin zum Skeletal Mesh zugreifen. Auch die Zusammenarbeit via Houdini ist mit dem CSV (Common Container Format) gewährleistet.

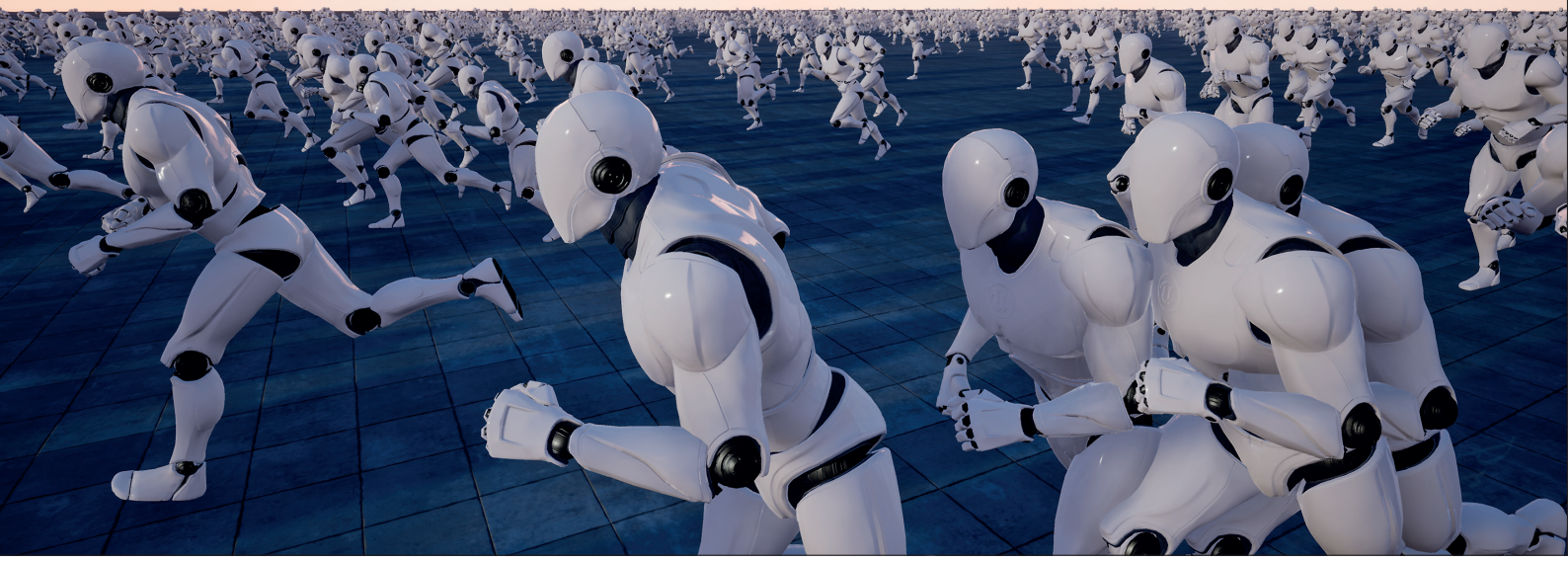
Niagara Highlights in UE 4.26

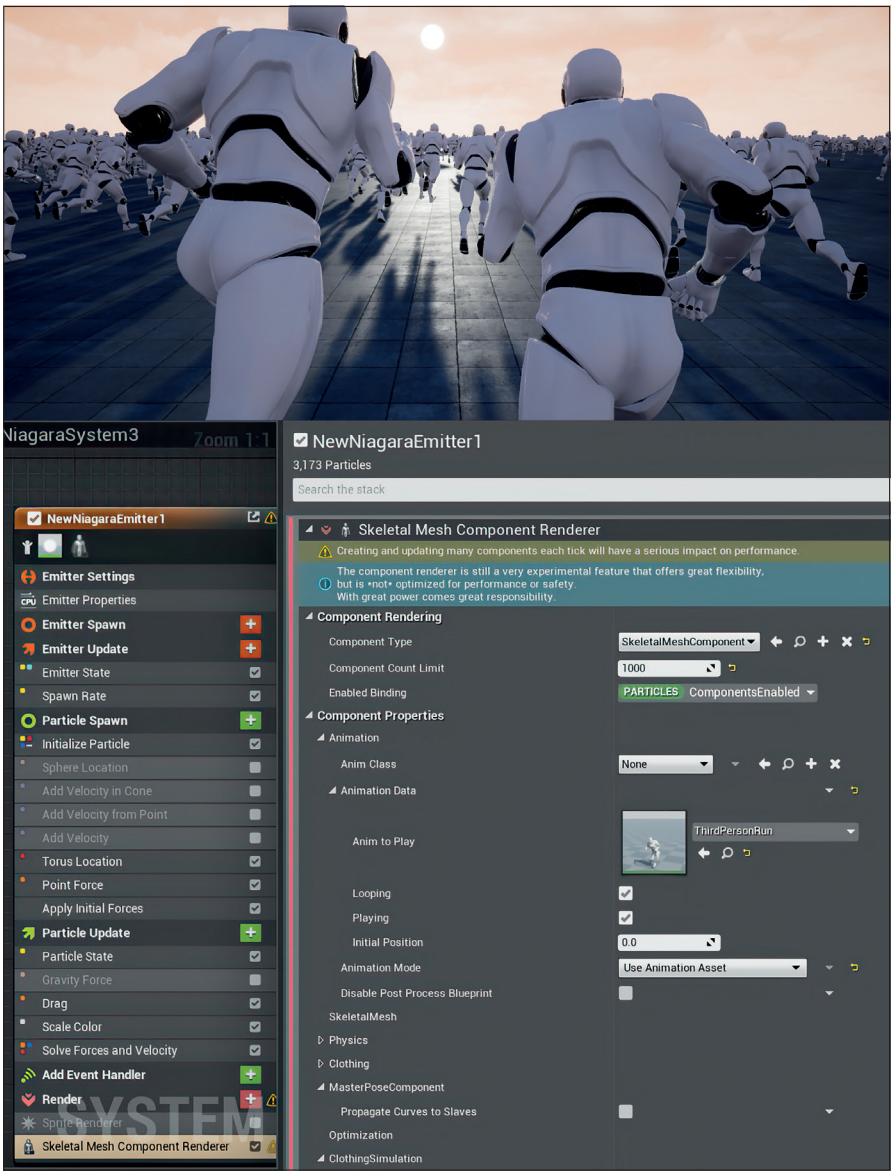
Neu in Unreal Version 4.26 ist der Niagara Component Renderer, der aber z.Z. noch experimentell ist. Dieses neue Feature ist in vielerlei Hinsicht ein Gamechanger, denn nun lassen sich erstmals Unreal Components als Niagara Particles benutzen, also praktisch alles, was auch einem normalen Blueprint Actor zur Verfügung steht. Das beinhaltet unter anderem auch Skeletal Animations, also animierte Meshes, die so problemlos in Crowd-Simulationen verwandelt werden können. Die Möglichkeiten, die dieses neue Feature eröffnet, sind atemberaubend.

Angefangen von massiven Zombiehorden, die nicht wie bei KI-Behavior mit rechenintensiven Bottlenecks und Komplexität belastet sind, bis hin zu verschachtelten Visual Effects, die jeweils in ihren eigenen Blueprints in Components genestet sind (via Child Component) und dann über Niagara gesteuert werden, ist alles machbar. Epic Games warnt aber, dass der Component Renderer noch hochgradig experimentell ist, was Implementierung und insbesondere auch Performance angeht. Die Standards bei Epic, was experimentellen Status betrifft, sind jedoch recht hoch, sodass diese in der Regel ziemlich stabil laufen. Nur sollte man sich natürlich in einer Produktion nicht auf ein experimentelles Feature verlassen.

Character-Schwarmverhalten im Component Renderer

Um den Component Renderer zu benutzen, fügt man ihn im Emittermodul hinzu, nachdem der Default Sprite Render deaktiviert wird. Im Selection Panel unter Component Type lässt sich nun jeder Component-Typ, den man von einem Blueprint Actor kennt, einfügen. Ein Skeletal Mesh Component zum Beispiel lässt sich mit den Component-typischen Parametern Animation und Skelett definieren und dann via Niagara-Parametern ansprechen, was insbesondere für die Rotations-





Mit dem Component Renderer werden Skeletal Mesh und Animation mit Partikelattributen versehen.

ausrichtung des Partikel-Characters notwendig ist. Wichtig ist, im Component Renderer den Component Count Limit auf die oberste gewünschte Grenze zu setzen, denn sonst greift die Spawn Rate nicht, falls die Werte darüber liegen. In einem kurzen Test war es kein Problem, den Level mit 800 bis 1.000 laufenden Mannequins aus dem UE4 Starter Content zu bevölkern, die via Point Force und Torus Location ausgeströmt wurden.

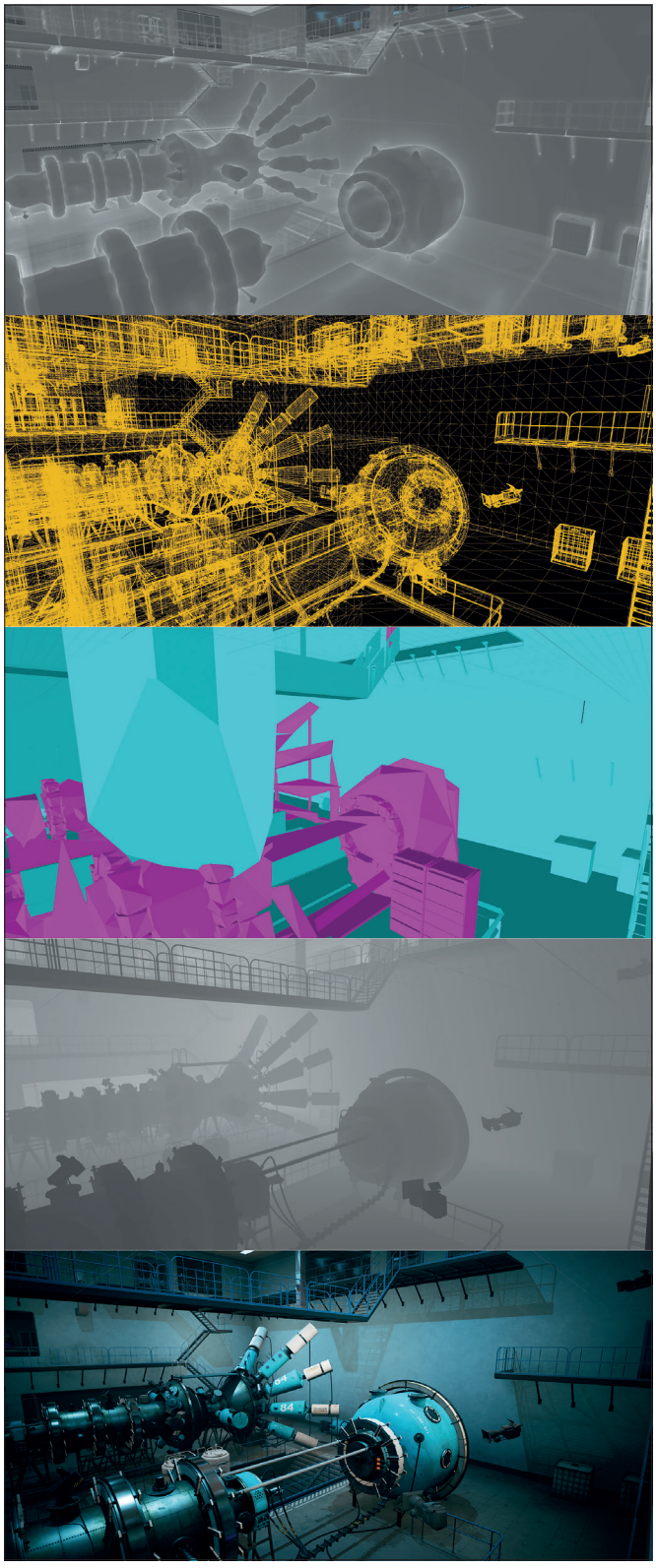
Viele UE4-Artists wurden von diesem neuen Feature überrascht, denn es waren ja insbesondere die Niagara Flocking Demos, die bei der Vorstellung des Unreal-5-Teasers für viel Aufmerksamkeit und Diskussion gesorgt hatten. Sowohl die Creature-Animationen der Niagara-Vogelschwärme als auch die lichtscheuen Insekten wurden dabei nicht skeletal ausgeführt, sondern über den Oldschool-Trick Vertexanimation, bei dem Vertex Offsets in Texturen eingebacken sind und so über Materialfunktionen animiert werden, da zu diesem Zeitpunkt skeletal Animation mit Niagara nicht möglich war. Vertexanimationen funktionieren gut, wie man es in der Unreal-5-Demo sieht, es gibt dazu ein Skript für 3ds Max und Blender, die Steuerungen darüber sind jedoch nur aufwendig zu erstellen. Für manchen Quickshot ist der Component Renderer mit geriggten, skeletal animierten Meshes da sicher eine gute und schnelle Alternative.

Niagara Performance und Profiling

Der wesentliche Vorteil von Niagara Particle Flocking, also Schwarmverhalten via Neighbor-Grid3D-Daten (als Spacial Hash), ist, dass es natürlich bedeutend performanter ist als z.B. eine KI-basierte Crowd-Simulation. Datenkommunikation via Particle Index oder Particle ID mit einem Attribute Reader ist in dieser Hinsicht wesentlich effektiver als ein eventbasiertes System.

Bei einem User Live Chat mit dem Dev Team von Niagara während des Releases wurde das Entwicklerteam deswegen auch gefragt, ob denn irgendwann einmal Skeletal Meshes via Niagara machbar sein würden. Vonseiten Epic hieß es dazu, dass es vielleicht in ferner Zukunft einmal dazu kommen wird, man könne aber nicht sagen wann. Dass nun „in ferner Zukunft" direkt „ein paar Monate später" heißt, ist daher überraschend und vielleicht ein Hinweis darauf, dass bei Epic

Der Niagara Profiler in UE 4.26 wird via Performance-Button aktiviert.



Vier Wege, über die Niagara eine Schnittstelle zur Game-Welt hat (von oben): Mesh Distance Fields, Triangles, Physics Volume und Scene Depth.

eventuell mit anderen Zeitmaßstäben gearbeitet wird.

Ein weiteres, wichtiges neues Feature in 4.26 ist die Möglichkeit, Audio von Partikelsimulationen auszuspielen. Dieses Feature macht Partikel-Audio-Synchronisation einfacher und wird auch Artists interessieren, die sich mit den Niagara-Realtime-Sound-Visualization-Modulen auseinandersetzen. Denn im Audiovisualisierungsbereich, der gerade für Live Performances gerne benutzt wird, gibt es ja bereits ein paar sehr nützliche Library Assets, die den Einstieg erleichtern.

Als letztes Highlight soll noch der neue Niagara Profiler erwähnt werden. Er ist elegant in das UI eingebettet, aber auch gleichzeitig mit dem bereits vorhandenen Stats-System integriert. Ein Knopfdruck genügt, und man hat in den Modulen eine dezente Komplettübersicht darüber, wo die Partikelkonstruktion gerade am teuersten ist. Viele kleine Verbesserungen sind in dem letzten Release natürlich auch mit dabei, so zum Beispiel neue Library-Module wie die neuen Mesh-Distance-Field-Skripte, die wir uns in einem kleinen Test anschauen können.

Mesh Distance Fields

Die Mesh Distance Field Information ist eine der vier Niagara World Interfaces in Unreal. Neben Mesh Triangles, Physics Volume und Scene Depth ist Distance Fields ein Weg, mit der Game-Welt zu interagieren, in diesem Fall über die Entfernung von der jeweiligen Mesh-Oberfläche.

Distance Fields sind also ein Werkzeug, um die Partikel dorthin zu bekommen, wo man sie gerne haben möchte, denn das ist neben dem Partikelverhalten und der Partikelerscheinung in der Regel das größte Problem. Distance Fields erlauben unter anderem die Möglichkeit, Partikel entlang einer Mesh-Oberfläche zu orientieren und zu bewegen. Es gibt natürlich unzählige Beispiele wo das hilfreich sein kann, angefangen von Insekten, die über ein Terrain ausströmen, bis hin zu der vorher erwähnten Character Disintegration bei Oberflächenannäherung. Das neue Modul Script liegt in der Emitter-Library bereit, wenn man via Rechtsklick mit der Suchfunktion Distance Fields eingibt.

Praxistest im Reaktorlabor

Um die Wechselwirkung von Distance Fields und Niagara zu testen, benutzen wir eine Szene aus dem Unreal Marketplace (Science Laboratory von SilverTM). Das beinhaltete Showcase Level bietet eine wohlige Half-Life-Atmo und lädt zum Experimentieren ein.

Drei Emitter mit Ribbon Renderer, um Blitze zu erzeugen.

Um die Module in der Praxis durchzutesten, legen wir uns auf eine spontane Idee fest: Ein Laborunfall durch Reaktorversagen manifestiert versehentlich eine Erscheinung aus dem Paralleluniversum, was ja immer gerne mal passiert.

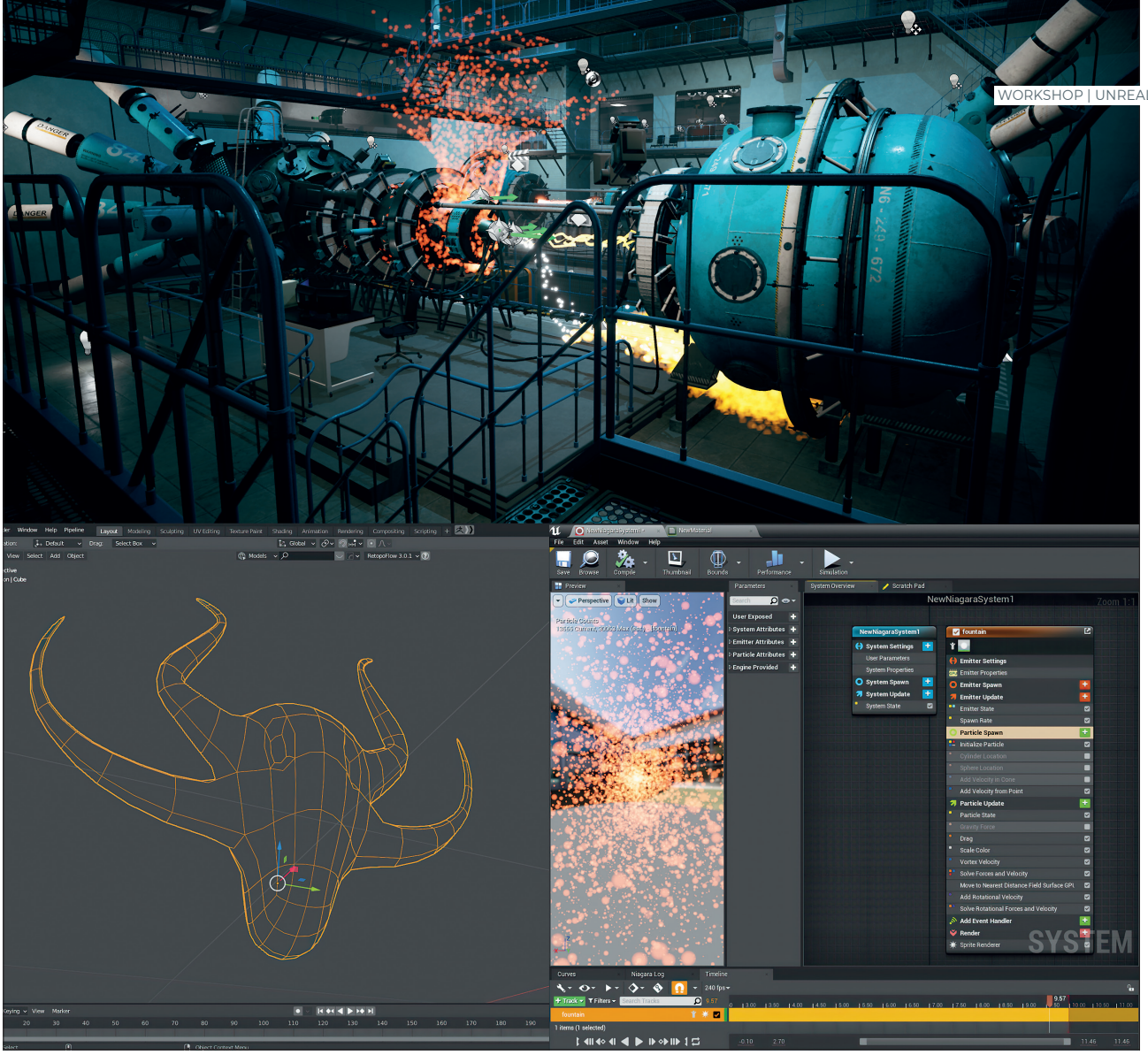
Wir wollen also die Fehlfunktion mit ein paar Emittern illustrieren und dann die Erscheinung mittels der Distance-Fields-Funktion visualisieren. Um die Funktion benutzen zu können, ist es jedoch zunächst notwendig, Distance Fields in den Unreal Project Settings zu aktivieren, was einen Neustart des Editors verlangt.

Für den Schnelleinstieg in Niagara reicht dann ein Rechtsklick im UE4 Content Browser, um dann via Create Advanced Asset > FX ein Niagara-System zu initialisieren. Das darauffolgende Fenster (Pick a starting point for your system) bietet 4 Möglichkeiten. Erstellt man ein leeres System, kann man diesem Emitter hinzufügen, die man entweder unabhängig im Content Browser via Rechtsklick oder direkt im Niagara-System erzeugt. Der einfachste Weg, einen vorher erzeugten Emitter hinzuzufügen, ist über das Track-Fenster, indem man ähnlich wie in einem Videoeditor die jeweiligen Emitter als Zeitleisten-Track hinzufügt. Emitter können unabhängig im Content Browser erstellt werden. Um aber im Game Environment platziert werden zu können, brauchen sie ein Niagara-System, denn nur das lässt sich per Drag-and-drop in der Game-Welt platzieren. Eine Übersicht der Funktionsweise erlaubt die Presets mit voreingestellten Emittern für den Schnellstart.

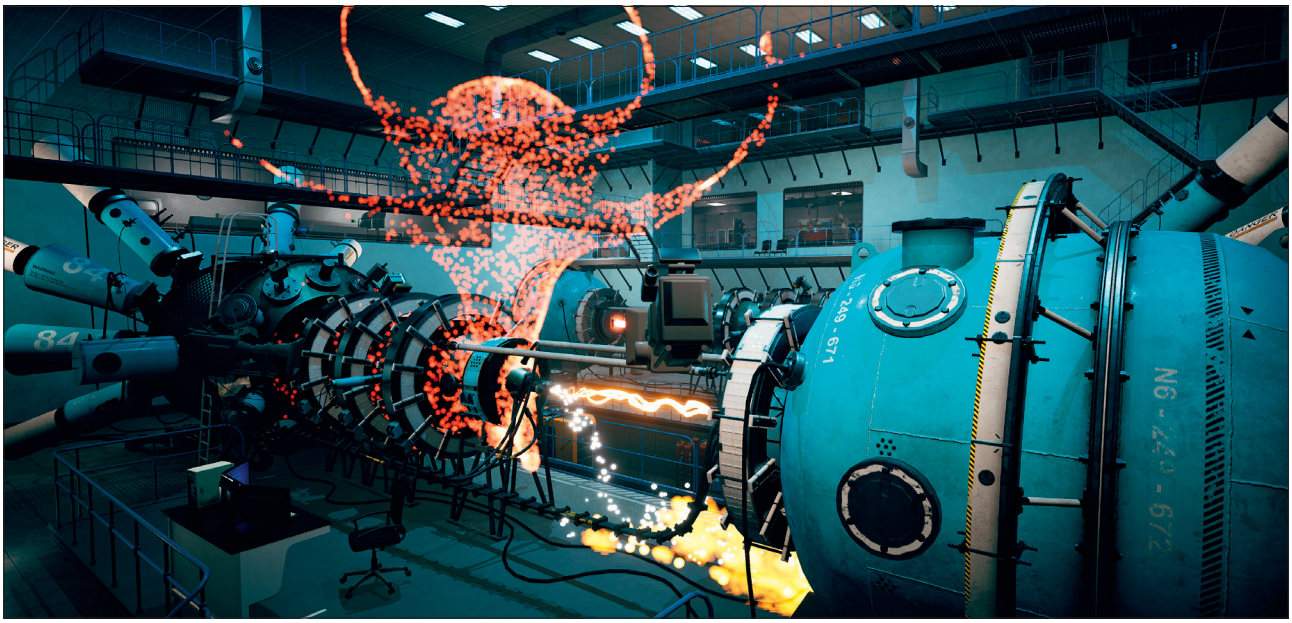
Blitze und Partikel mit Kollision

Zunächst einmal werden wir mit Blitzen und Leuchtpartikelausstoß darauf hinweisen, wo das Missgeschick seinen Ursprung hat. Dazu wird der ursprünglich in der Szene benutzte Cascade Effect via Outliner unsichtbar gemacht, denn dieser soll nun durch ein neues Niagara-System ersetzt werden.

Wir erstellen ein Niagara-System mit elektrischen Blitzen und benutzen dazu das Niagara Static Beam Preset mit einem Ribbon Renderer im Emitter-Modul, denn dieser hat den alten Beam von Cascade abgelöst. Der Ribbon Renderer verbindet Partikelpunkte über einen Streifen, d.h. je höher der Spawn Count ist, desto höher sind die Unterteilungen des Streifens in Segmente.



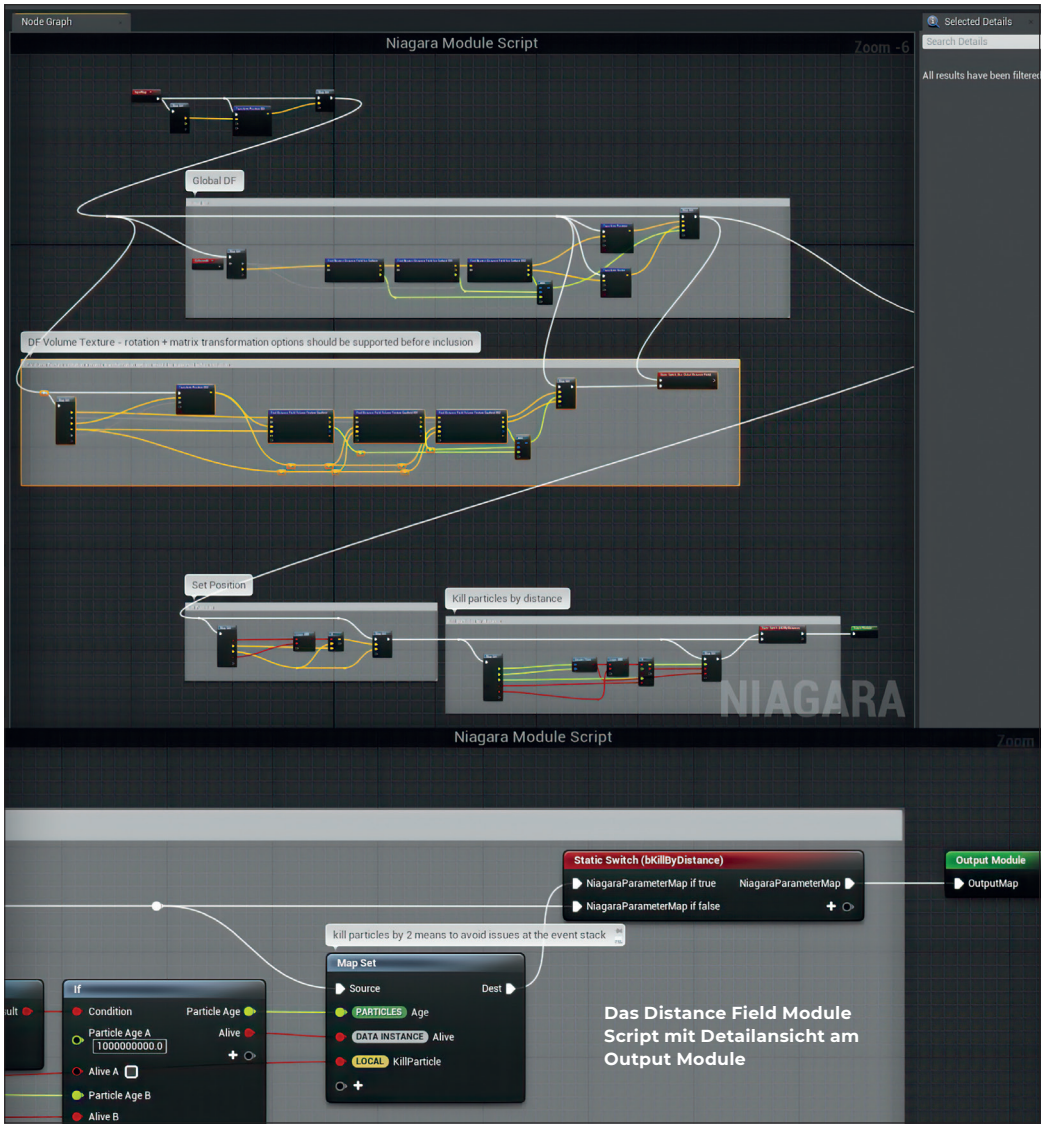
Das Blender Mesh wird via FBX in die Szene eingesetzt, einem Niagara-Emitter mit Distance Field Script eingepflanzt und zuerst mit 0,3 Opacity, dann später mit 0,0 Opacity unsichtbar gemacht.



Das Mesh fängt die Distance-Field-Partikel auf und gewinnt so an Kontur.

Weil wir kantige Zuckungen erzeugen wollen, ist daher ein niedriger Spawn Count naheliegend. Um die gewünschten Bewegungen zu erzeugen, reicht lediglich ein Jitter-Position-Modul, das nach einigen Tweaks dem Blitzeffekt nahe genug kommt, nachdem wir auch die Lichtstärke-Values im Farbraum um 500 Prozent nach oben gedreht haben.

Um das Erscheinungsbild komplexer und überzeugender zu machen, duplizieren wir den Emitter via Copy-and-paste zwei weitere Male. Im Grunde ist dies, wie jeder, der mit Game Effects zu tun hat, weiß, schon die halbe Miete, denn häufig reicht es, mehrere gleiche Emitter übereinanderzulegen, um via leichten Veränderungen in jedem Modul das komplexe Zusammenspiel hinzubekommen.



Das Distance Field Module Script mit Detailansicht am Output Module

Das nächste Niagara-System, das wir in die Szene einfügen, beinhaltet das Fountain-Preset-Modul. Wir möchten nun erreichen, dass der funkenartige Partikelausstoß via Kollision auf dem Boden abprallt und bei Bodenkontakt eine Art Dampf auslöst. Dazu müssen wir zwei Emitter via Events miteinander verknüpfen.

Niagara Events

Um Niagara-Partikelevents nutzen zu können, müssen zunächst die Persistent Particle IDs im Niagara-System freigegeben werden, denn jeder einzelne Partikel benötigt seine eigene ID, um via Event angesprochen werden zu können. Persistent ID ist aus Performancegründen per Default ausgestellt, lässt sich aber im Niagara-System aktivieren. Für das Aufprall-Event brauchen wir zuallererst das Collision-Modul als Basis. Es sorgt dafür, dass die Partikel an der nächsten Oberfläche abprallen, so wie es für Funkensprung üblich ist.

Bevor wir den ersten Fountain-Emitter duplizieren, fügen wir dann ein Generate-Collision-Event-Modul ein. Als Events stehen Birth, Collision und Death zu Verfügung. Das nun erzeugte Collision-Event für jeden einzelnen Partikel kann nun in einem anderen Emitter abgerufen werden. In unserem Fall soll der dichte Bodendampf bei der Bodenkollision erzeugt werden, also wird bei dem zweiten entsprechend optisch angepassten Emitter dann das Event Handler aufgerufen. Aus dem Event-Handler-Pulldown-Menü lässt sich nun recht einfach das vorher aufgerufene Event auswählen, im zweiten Pulldown-Menü Spawned-Partikel einstellen und etwas darunter die Spawn Number justieren.

Nun fehlt nur noch das Receive-Collision-Event-Modul, damit die ganze Sache auch funktioniert. Bei der Simulation wird nun der zweite Emitter bei Partikel-Bodenkollision des ersten Emitters ausgeführt.

Durch diese Verknüpfungen zwischen Emittern sind viele interessante Effekte mit sekundären Dynamiken erreichbar. Auf einem Epic Event wurde im Entwicklerteam auch darüber gesprochen, dass man eventuell in Zukunft erwägt, die Emitterverknüpfung auch visuell anzuzeigen, wie es bei Blueprint Nodes der Fall ist. Solange die Emitteranzahl überschaubar ist, geht es aber auch problemlos ohne.

Das Distance Fields Module Script

Das dritte Niagara-System erzeugen wir mit einem Sprite-basierten GPU-Emitter, der via Velocity from Point ausströmt. Als Mesh bauen wir schnell in Blender einen gehörnten Kopf zusammen, der via FBX importiert und in die Szene eingefügt wird. Um dann die Auswirkungen auch innerhalb des Meshes bewundern zu können, wird ein transparentes Material aus dem Material Editor benötigt. Nun brauchen wir lediglich das Emitter-Script-Modul „Move to Nearest Distance Field Surface GPU" in den Emitter einzufügen. Das Niagara-System schlägt nun unmittelbar auf die nächste Oberfläche an, was heißt, die Oberflächen ziehen Partikel an und diese verbreiten sich dann mit Haftung an diesen. Wenn nun das unsichtbare Mesh in die Nähe des Emitters gezogen wird, wachsen die Partikel in die Form hinein und bilden so die Silhouette im Laufe der Simulation.

An diesem einfachen Beispiel lässt sich zeigen wie intuitiv und spielerisch mit Distance Fields umgegangen werden kann. Denn anders als bei anderen Ansätzen lassen sich Geometrien ineinanderschieben und animieren, d.h. der Emitter sucht sich immer die nächstgelegene Oberfläche aus und wirkt dabei immer organisch und natürlich. Es eröffnen sich viele Möglichkeiten, um mit unsichtbaren Meshes Partikel zu führen, auch wenn diese zum Beispiel über den UE4-Sequencer animiert werden. Um dann den entsprechenden Look zu entwickeln, ist dies natürlich nur der Anfangspunkt, denn nun kann ja auch das Distance-Field-Scrip-Modul modifiziert werden, zum Beispiel mit Offset-Werten oder Ähnlichem. Das alles sind Möglichkeiten, die bei Cascade undenkbar waren, zumindest ohne Programmiererteam im Rücken.

Niagara: Ein gelungener Start in eine neue VFX-Generation

Niagara ist noch recht neu, und auch erfahrene Unreal-User tasten sich gerade erst an das Potenzial heran. Was aber klar ist: Es handelt sich hier um eine VFX-Bestie, die gezähmt werden will. Epic Games hat gezeigt, dass neben Datenschnittstellen, modularer Struktur und Graph-Paradigmen UX bzw. Benutzerfreundlichkeit hohe Priorität bei der Entwicklung hat. Wenn es in diesem Tempo weitergeht, kommt sicher noch einiges auf die VFX-Welt zu. **› ei**

Cornel Hillmann ist CG-Artist mit Sitz in Singapur (**studio.cgartist.com**) und Autor des Buches „Unreal for Mobile and Standalone VR" des New Yorker Apress Verlages.