

## **Houdini Introduction**

-

## **Prozedurale Modellierung**

von

Jana Koch, Laura Wagner,

Nedim Thull, Ali Said, Samantha Maaf

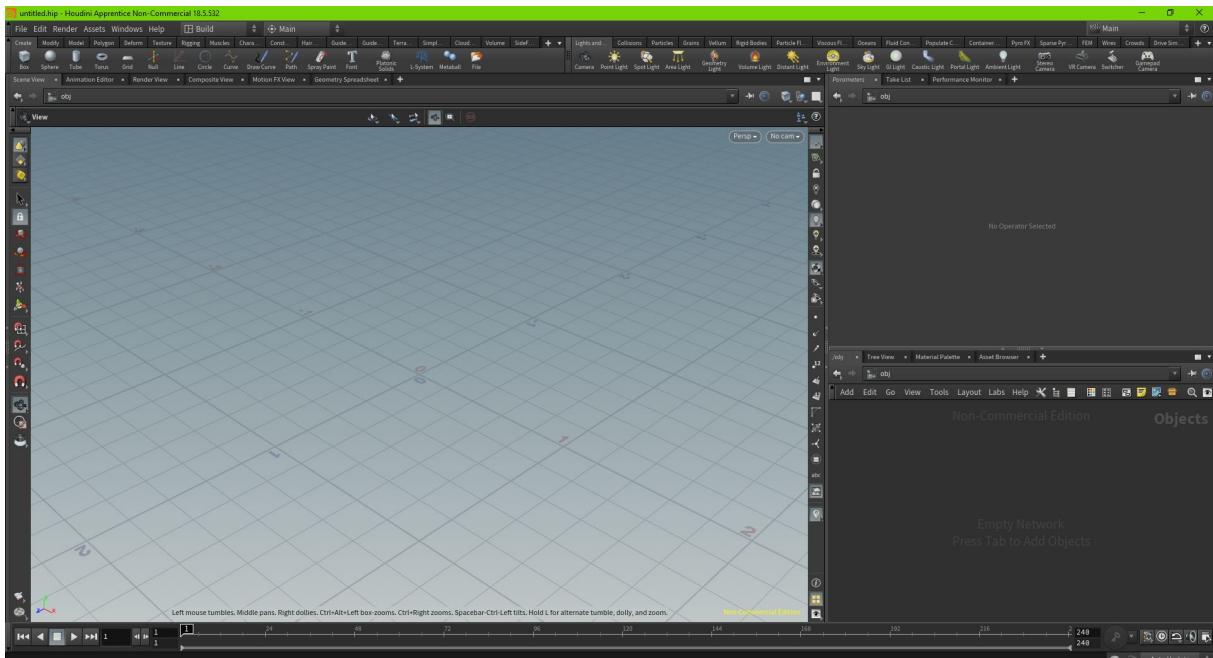
## Getting started:

- Download der Free Test Version von Houdini 18.5.532 from <https://sidefx.com/download/>.
- Registrierung mit Username und E-Mail notwendig.
- Anmeldung auf der Webseite bevor Download möglich ist.
- Bei der Installation ist es ratsam, neben den bereits markierten Komponenten, direkt die 'SideFX Labs' zu installieren. Dies ist kein muss, spart allerdings Zeit falls man sie später doch benötigt. SideFX Labs ist ein Toolset zur Unterstützung von Aufgaben bei der Erstellung digitaler Inhalte.
- Beim erstmaligen starten von Houdini muss eine Lizenz installiert werden, in unserem Fall die 'free Houdini Apprentice license'.

## Was ist Houdini?

Houdini ist eine 3D-Grafiksoftware, deren Hauptaugenmerk auf der prozeduralen Modellierung liegt. Prozedurale Modellierung beschreibt mit Regeln, wie ein 3D-Objekt erzeugt werden soll. Sie können durch Attribute gesteuert werden und beschreiben Transformationen programmatisch. Dies wird in Houdini mit 'Nodes' realisiert. Sie beschreiben Aktionen die auf die Objekte angewendet werden und können miteinander zu einem 'Network' verknüpft werden. Dadurch lassen sich schnell sehr ähnliche aber einzigartige Objekte schaffen und immer wieder anpassen.

## Interface:



Das Standard-Interface ist in drei Teilbereiche aufgeteilt:

- Die Scene View, eine 3D Ansicht der aktuellen Szene. (links)

- Der Parameter Editor, zum anpassen der Parameter der aktuell ausgewählten Node. (rechts oben)
- Der Network Editor, er zeigt alle Nodes des aktuellen Networks, man kann neue anlegen, sie miteinander verbinden und auswählen um die Parameter im Parameter Editor anzupassen. (rechts unten)

Die Panels können beliebig angepasst, und die Einstellungen gespeichert werden. Die Ansicht in der Scene View kann mit der Maus gesteuert werden: Linke Maustaste zum drehen, mittlere Maustaste zum verschieben und die rechte Maustaste oder das Mausrad zum zoomen. Es macht Sinn sich als erstes mit dem Interface vertraut zu machen, dazu haben wir uns hauptsächlich YouTube-Videos angeschaut. Besonders die 'Houdini isn't scary' - Tutorial Reihe von Nine Between (<https://youtu.be/Tsv8UGqDibc>) und die 'Getting started with Houdini' - Reihe von Arise.Works (<http://y2u.be/axD1ejlaEBA>) sind sehr zu empfehlen. Allerdings ist die Dokumentation von Houdini und die Tutorials von SideFX selbst sehr gut. (<https://www.sidefx.com/docs/houdini/>)

## Tipps we wish we knew before:

- Mit 'Shift + Space + H' wird die Scene View wieder zurückgesetzt.
- Der Desktop kann komplett auf Standard zurück gesetzt werden, indem er neu geladen wird.
- Nodes können gruppiert, eingefärbt und in Subnetze unterteilt werden.
- 'Alle Wege führen nach Rom', aber es gibt immer einen Weg etwas schneller zu machen.
- Speichern. Oft.
- ESC spammnen falls Attribute zu hoch - oder zu niedrig - gesetzt wurden kann einen Programmabsturz verhindern.
- Man kann im Parameter Editor die Node nach 'geänderten Parametern' durchsuchen: Lupe > Search All -> 'Parameters with Non-Default Values'
- Man kann die Panels auf mehrere Bildschirme verteilen: Pfeil in der rechten oberen Ecke des Panels > Tear off Pane Tab.

## Nodes:

### Attribute:

#### **Attribute Blur:**

Verwischt Gleitkommaattribute, einschließlich Position, Farbe und Texturkoordinaten, von Punkten in einem Netz, wodurch eine Geometrie effektiv geglättet wird.

- Attributes - Welche Attribute verwischt werden sollen, wenn das Feld leer ist, macht die Node nichts

- Method
  - Uniform - verwischt gleichmäßig
  - Edge Length - verwischt mit Wissen das manche Punkte näher sind als andere und behält ihre Abstände bei
- Mode
  - Laplacian - jede Iteration hat dieselbe Größe die durch 'step size' definiert wird
  - Volume Preserving - Iterationsgrößen werden basierend auf einer Rauschfrequenz ausgewählt, die durch 'Cutoff Frequency' ausgewählt wird
  - Custom Benutzer - kann die Iteration Größen von geraden und ungeraden manuell angeben
- Influence Type
  - Connectivity - Punkte mischen ihre Attributwerte mit den ihrer Nachbarn, die durch Mesh-Konnektivität (Verbindungsfähigkeit) erkannt werden
  - Proximity - Nachbarn werden durch die Nähe bestimmt
    - \* Max - max. Anzahl der Nachbarn
    - \* Proximity Radius - max. Entfernung der Nachbarn

### ***Attribute create:***

Erstellt Attribute der Typen float, Integer, Vector oder String. Falls kein Variablen Name genannt wird, wird der Attribute Name genommen (Groß Buchstaben).

Value: Stamp () gibt eine stamping Variable eine meist danach liegenden Copy Node zurück

### ***Attribute transfer***

Überträgt Scheitelpunkt-, Punkt-, Grundelement- und/oder Detailattribute zwischen zwei Modellen. Die Übertragung funktioniert durch Nähe und überträgt Attribute von einem Geometrie Teil auf die nächstliegende Punkte auf einem anderen Geometrieteil.

unter Attribute kann man die jeweiligen Attribute festlegen. Wenn das Feld leer oder ein Sternchen beinhaltet werden alle verfügbaren Attribute übertragen.

- Conditions - gibt an wie Quellattribute kombiniert oder gefiltert werden
- Distance Threshold- gibt die maximale Entfernung von Quellpunkten/Primitiven an, die berücksichtigt werden müssen
- Blend Width - lockert den Distance Treshold und legt optional einen Bereich außerhalb des Distance Treshold fest, in dem der Einfluss der Quellattribute allmählich nachlässt.

### **Attribute Randomize:**

Diese Node generiert Zufallswerte, um ein Attribut zu erstellen oder zu ändern. Beispiel Parameter:

- Gruppe – Die Teilmenge der Geometrie, deren Attribut geändert werden soll.
- Attributklasse - Der Elementtyp, für den das durch den Attributnamen angegebene Attribut erstellt oder geändert werden soll.
- Gruppentyp - Der Gruppentyp, den Group angibt.

### **Attribute VOP:**

Innerhalb der Node (Doppelklick auf die Node) befindet sich ein VOP Netzwerk. Mit Hilfe von Mathematischen Nodes (z.B. Multiply) lassen sich Attribute Modifizieren. genutzte Nodes:

- geometryvopglobal alle globalen Variablen für die Attribut-VOP-Netzwerktypen
  - p - Position des aktuellen Elements
  - ptnum (pointnumber) - aktuell verarbeiteter Punkt
  - numpt (number points) - Gesamtanzahl der Punkte
- divconst - dividiert eingehenden wert durch angegebenen konstanten Wert (Divider)
- divide - Division jedes Eingabewerts durch den nächsten
- multiply - gibt das Produkt der Inputs aus
- inttofloat - wandelt integer zu float um
- vectofloat - wandelt einen Vector zu einer float Zahl um
- floattovec - wandelt eine float Zahl zu einem Vector um
- bind - Bindet die Geometrie an die VEX Funktion um Attribute aufrufe zu können
- ramp- ramp user interface. Der Input ist die Position in der Rampe von dem Wert, welche die Node ausgibt
- turbnoise - berechnet rauschen mit der Fähigkeit, Turbulenzen mit Rauheit und Dämpfung zu berechnen
- displaceNormal - Verschiebt die Fläche entlang der Flächennormalen um einen bestimmten Betrag
- geometryvopoutput - Platzhalter, um Attribut-VOP-Netzwerktypen herauszuschreiben

### **Attribute wrangle:**

entspricht der attribvop Node, aber anstatt des VOP Netzwerks werden hier VEX code Schnipsel genutzt. VEX(Vector Expression Language) ist die Coding Sprache die inner-

halb Houdini Verwendet wird und auf C/C++ basiert. Innerhalb des VEXpressions Fenster kann gecodet werden. Verwendeter VEX Code:

@ repräsentiert Attribute:

- @Cd - colour (diffuse)
- @ptnum - aktuell verarbeiteter Punkt
- @pscale - steuert die Größe (Skalierung) der Punkte
- @elemnum - Nummer des Aktuellen Elements
- @numpt - Gesamtanzahl der Punkte
- @np - die Input Nummer
- @rot - Rotation
- @sc - Skalierung
- @Time - float Time(\$T)

Der Buchstabe vor dem @ repräsentiert den Datentyp:

- f@name - float
- i@name - integer

Funktionen:

- addpoint() - int addpoint(int geohandle, int point\_number)  
fügt Punkte einer Geometrie dazu geohandle Geometrie, in die der Punkt hinzugefügt werden soll momentan ist der einzige gültige Wert 0 (oder geoself), was die aktuelle Geometrie in der Node bedeutet
- setpointattrib() - int setpointattrib(int geohandle, string name, int point\_num, <type>value, string mode='set')  
Legt ein Punktattribut in einer Geometrie fest:
  - name - Attribut welches festgelegt werden soll
  - point\_num - Nummer des Punktes auf dem das Attribut gesetzt wird
  - value - Wert des Attributes
  - mode - (Optional) wenn angegeben, steuert wie die Funktion jeden vorhanden Wert im Attribut ändert 'set' überschreibt das Attribut mit dem angegebenen Wert
- ch() - float chf(string channel): Wertet einen Channel oder Parameter aus und gibt seinen Wert zurück. Buchstabe nach dem ch gibt Datentyp an:  
f -> float  
ramp -> ramp  
float -> chramp(string channel, float ramppos): wertet einen Ramp Parameter aus und gibt seinen Wert zurück  
ramppos -> stelle auf der Rampe, die ausgewertet werden soll  
mit ch() erstellte kann man durch klicken von 'Creates spare parameters for each'

unique call of ch()' anzeigen lassen und bearbeiten.



- noise() - Es gibt zwei Arten von noise: ändert sich zufällig im gesamtem N-dimensionalen Raum  
nicht periodisches noise: wiederholt sich in einem gegebenen Zeitraum  
Die Funktion bestimmt die Art:  
vector/float noise(float pos) -> 1D noise  
vector/float noise(float posx, float posy) -> 2D noise  
vector/float noise(vector pos) -> 3D noise  
vector/float noise(vector4 pos) -> 4D noise  
man bekommt entweder einen Vector aus 3 Zahlen oder eine zufällige float

## Digital Assets:

### ***Subnet:***

Die Subnet Node kann, wie der Name schon sagt, verwendet werden, um eine Menge an Knoten in einem Subnetz zu gruppieren. Man kann dann durch rechtsklick auf das Subnetz > 'Create Digital Asset', ein digital Asset erstellen. Diese werden in einer eigenen Datei mit .hda - extension gespeichert. Man kann damit eigene Netzwerke in eine personalisierte Node mit eigenem Interface erstellen. Bei Rechtsklick auf das Subnetz 'Parameters and Channels' > 'Edit Parameter Interface', öffnet sich eine Übersicht der existierenden Parameter. Dort können die gewünschten Parameter der gruppierten Nodes per 'Drag and Drop' hinzugefügt werden.

## Group:

### ***Group Create:***

Erzeugt Gruppen von Punkten, Primitiven, Kanten oder Scheitelpunkten nach verschiedenen Kriterien.

- Base Group:
  - Base Group - gruppierende Muster, behandelt normale adhoc-Gruppen (keine dauerhafte Gruppe, nur unmittelbar zur Lösung eines bestimmten Problems gebildet) Syntax
  - create ordered - ordnet die Gruppe
- Keep in Bounding Regions:
  - Enable - Gruppieren nach begrenzungsvolumen

- Bounding Type - Form des begrenzungsvolumen
- Size - Größe des begrenzungsvolumen
- Center - Mitte des begrenzungsvolumen
- Iso - Iso Fläche des Volumens, welche Gruppiert werden soll, Punkte mit niedrigerem Volumenwert werden gruppiert.
- Invert Volume - Volumenwert nicht mehr niedriger, sondern größer

## **Import:**

### ***Objekt Merge:***

Ermöglicht das zusammenführen mehrerer Geometrien. Dadurch reicht es eine Blüte zu erstellen, die wir in allen Blumen Verwenden Können. Wir referenzieren dafür auf die Null-Node, die am Ende jeder Geometrie liegt.

Objekt 1 -> klicke auf 'Open floating operaor Chooser' -> wähle das Objekt (Null-Nodes)

## **Labs > WorldBuilding > Tree:**

### ***Tree \_ Trunk \_ Generator:***

Die Tree Trunk Generator Node ist eine einfache Node, um einen Baumstamm automatisiert erstellen zu können. Hier kann man einstellen, wie lang der Stamm sein soll und wie groß der Radius sein soll.

### ***Tree \_ Branch \_ Generator:***

Bei dieser Node ist der Aufbau und die Funktion komplett gleich wie bei dem Tree \_ Trunk \_ Generator. Was man aber hier anders einstellen kann, ist wie die Gravitation wirken soll. Ob die Äste nach oben, nach unten oder in eine bestimmte Richtung stehen sollen.

### ***Tree \_ Leaf \_ Generator:***

Mit Hilfe dieser Node kann man die Punkte einstellen, an denen sich die Blätter am Ast befinden sollen. Man kann einstellen, wo genau die Blätter sein sollen und wie viele es sein sollen.

## **Manipulate:**

### ***Bend:***

Mit der Transformation Node lässt sich das Objekt verschieben, vergrößern, skalieren und rotieren. Bend kann es dazu noch verformen und zum Beispiel biegen und verdrehen. Durch Setzen des Haken bei 'Deform in Both Directions' wird die Verformung in beide

Richtungen angewendet, welche als Default nur in eine Richtung geht. Um eine Verformung nutzen zu können, muss diese einfach nur aktiviert werden.

Möglichen Verformungen:

- Bend - Biegen
- Twist - Verdrehen
- Length Scale - strecken(+) und stauchen(-)
- Taper- Skalieren am Mittelpunkt

### ***Mountain:***

Verschiebt Punkte entlang ihrer Normals basierend auf noise. Sie wird häufig verwendet um eine Ebene zufällig zu deformieren. Dies kann auf ein ganzes Objekt oder nur eine bestimmte Gruppe angewendet werden. Zu den wichtigsten Parametern gehören:

- height -> die Höhe der Verschiebung, also wie weit die Punkte von dem eigentlichen Objekt maximal entfernt sein dürfen
- Element Size -> Die Distanz zwischen Spitzen der minimalen Noise
- roughness -> Je höher der Wert, desto 'Spitzer' wird das Ergebnis.

### ***Transform:***

Mit der Transform Node, können Objekte beliebig transformiert werden. Dazu gehört das verschieben an eine beliebige Stelle der Szene, sowie das verkleinern, vergrößern oder rotieren in alle Richtungen.

### ***Material:***

#### ***Color:***

Mit der Color-Node kann man ein Objekt einfärben.

### ***NURBS:***

#### ***Skin:***

setzt eine Haut auf eine Form, die eine Oberfläche definiert. Ebenso kann man auch zwischen zwei Oberflächen eine Haut erstellen.

Die Anzahl der eingehenden Objekte definiert dabei die Skinning-Methode. Bei nur einem Objekt wird 'Linear-skinning' verwendet, welches eine Haut über Querschnitte zieht. Bei mehreren eingehenden Objekt wird 'Vilineares Skinning' durchgeführt, welches eine Haut zwischen den Objekten zieht.

## **Particle:**

### **Scatter:**

Diese Node verteilt neue Punkte in einem ungefähr gleichförmigen Muster über die Oberfläche und versucht optional, Klumpen und Löcher zu begrenzen. Bei Volumenprimitiven streut Scatter Punkte durch das Volumen mit einer Dichte proportional zum Feldwert (wobei negative Werte eine Wahrscheinlichkeit von Null ergeben).

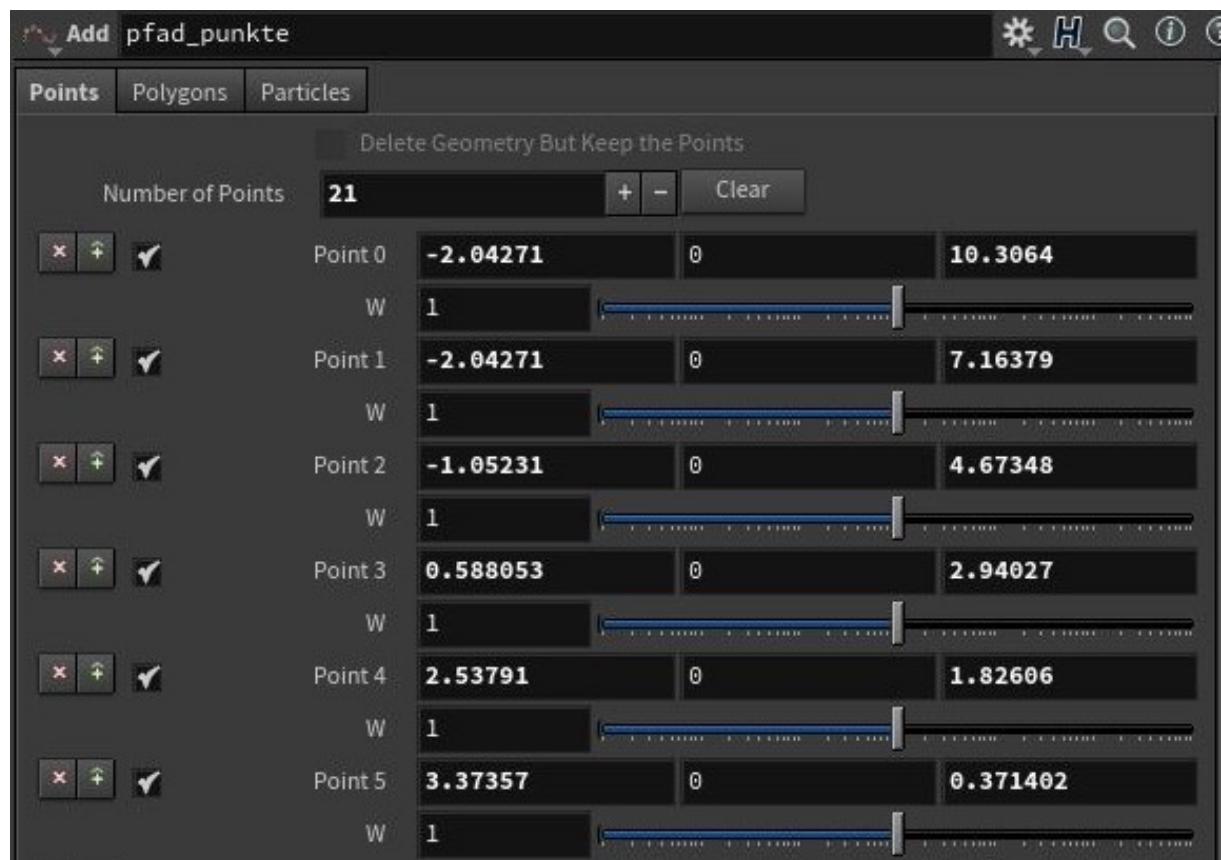
## **Polygon:**

### **Add:**

Mit add kann man entweder Punkte oder ein Polygon erstellen, oder man fügt Punkte/-Polygone einem Input hinzu.

Points:

- Number of Points - Anzahl der Punkte
- PointX - die drei Felder entsprechen den x,y,z coordinate
- w - weight (Gewicht), wenn die Punkte später zu einer spline (NURBS or Bezier) werden, beeinflusst das Gewicht die Form und kann dazu führen das sie rational wird



## **Boolean:**

Kombiniert entweder zwei Objekte mit Boolean Operatoren oder findet ihre Schnittlinie.

Set Treat As Solid -> Festes Objekt

Surface -> Fläche ohne innen und Außenseite

Funktionen:

- Set A -> Solid, Set B -> Solid
  - Union - neues Festes Objekt mit dem Volumen von beiden gegebenen Objekten
  - intersect - neues Festes Objekt mit dem gemeinsam genutzten Volumen der gegebenen Objekte
  - subtract - entfernt das geteilte Volumen von einem oder beiden gegebenen Objekten
  - shatter - Kombination aus Intersect und Subtract: schneidet entlang der Schnitstellen, um neue Objekte zu erstellen
  - seam - gibt Polylinien aus, an denen sich die Oberflächen schneiden
- Set A -> Solid, Set B -> surface
  - Union - kombiniert ein festes Objekt mit doppelten Wänden um Teile der Fläche außerhalb des Volumens des Festen Objekts herum
  - intersect - schneidet alle Parts der Fläche weg die außerhalb des festen Objekts liegen
  - subtract - beim Subtrahieren einer Fläche von einem festen Objekt entstehen doppelwandige Schnitte im festen Objekt, wo er die Objekte schneidet
  - shatter - Kombination aus Intersect und Subtract: schneidet die Schnittpunkte in eine neue Form
  - seam - gibt Polylinien aus, wo die Fläche den Festen Objekt schneidet
- Set A -> surface, Set B -> Solid
  - Union - kombiniert zusammentreffende Flächen und deren schneidende Teile der beiden Flächen
  - intersect - behält nur zusammentreffende Flächen
  - subtract - entfernt zusammentreffende Flächen und erstellt doppelseitige Schnitte entlang der Schnittpunkte
  - shatter - Kombination aus Intersect und Subtract: schneidet Schnittpunkte heraus und behält zusammentreffende Flächen
  - seam - gibt Polygone für zusammentreffende Flächen und Polylinien an den Schnittpunkten aus
- custom - Bei Festen Objekten mit überlappenden oder konzentrischen Flächen kann man Custom nutzen, um einen Bereich an einer gewissen Tiefe auszuschneiden

- detect - durchläuft die ersten Geometrien und fügt optionale Gruppen und/oder Attribute hinzu, die die sich schneidende Polygone enthalten.

### ***Facet:***

Mit Facet kann man die Geometrie in Etappen ändern. Man kann z.B. Oberflächennormalen berechnen, bevor man Punkte, die von verschiedenen Polygonen geteilt werden, eindeutig macht. Das führt zum ungewöhnlichen Ergebnis einer glatten Schattierung und eines eindeutigen Punkts, da die Normalen berechnet werden, während die Punkte noch geteilt werden.

### ***Normal:***

Dieser Knoten berechnet Punkt-, Scheitelpunkt-, Grundelement- oder Detailnormalen unter Verwendung eines genaueren Ansatzes als der Facettenknoten oder der Scheitelpunktknoten. Wird der 'Cusp Angle' auf 0 gesetzt, können die Kanten eckig gemacht werden.

### ***PolyBevel:***

Polybevel fügt Verrundungspolygone zwischen Kanten und in Ecken ein, mit großer Kontrolle über die Form der Verrundung, durch die Anzahl an Divisions und der Distanz. Diese Node kann sehr komplexe Eingaben verarbeiten und ignoriert nicht-beitragende Kanten.

### ***PolyExtrude:***

Mit Poly Extrude kann man Flächen und Kanten 'herausziehen', um neue Spalten/Blätter von Polygonen zu erstellen. Man kann die Extrusion vergrößern oder verkleinern (Einsatz/Anfang) und verdrehen.

### ***PolyReduce:***

Polyreduce bietet eine sehr schnelle, hochpräzise Reduktion, während Form, Texturen, Attribute und Quad-Topologie der Eingabe so weit wie möglich beibehalten werden. Es hat mehrere Funktionen, mit denen Sie kontrollieren können, wo der Knoten verkleinert und umgeformt wird:

- Sie können verhindern, dass der Knoten ungeteilte Kanten im 3D- und UV- Raum verschiebt.
- Sie können Punkte und/oder Kanten angeben, die beibehalten werden sollen.
- Sie können ein Attribut in Bereichen zeichnen, in denen Sie mehr Dichte beibehalten möchten.
- Sie können Polygone basierend auf der Sichtbarkeit von bestimmten Ansichtspunkten aus beibehalten.

### ***Remesh:***

Rekreiert die Fläche mit 'hochwertigen' (fast gleichseitigen) Dreiecken (hochwertiges dreiecksnetz = alle Winkel so nah wie möglich an 60 Grad). Dabei wird versucht in jedem Dreieck den kleinsten Winkel zu maximieren. Es gibt zwei Arten von Remeshing:

- Uniform - versucht Kantenlängen auszugleichen und erstellt so gleichgroße Dreiecke
- Adaptive - passt die Größe den Bereichen an und erstellt größere Dreiecke in breiten Bereichen und kleinere Dreiecke in detaillierten Bereichen
- Iterations - die Qualität des Netzes, Im Normalfall ist die maximal nützliche Anzahl 3 bis 4
- Recompute Normals - berechnet neue normals für das generierte netz
- Smoothing - grad der Glättung

### ***Subdivide:***

nimmt eine Polygon Fläche und teilt jeden Bereich, um die Fläche zu glätten.

- Groups - Alle Polygone in der linken Eingabe, werden verwendet, um das zu unterteilende Polygonnetz zu bestimmen.
- Creases - Elemente der rechten Eingabe werden als Knicke genutzt.
- Depth - Wie viele Iterationen zu unterteilen sind

### ***Primitive:***

#### ***Box:***

Ein einfaches/-r Quadrat/Quader.

#### ***Grid:***

Erstellt eine flache Ebene, die aus einem Netz, Bezier- und NURBS-Flächen oder mehreren Linien mit offenen Polygonen bestehen kann. Häufig genutzte Parameter:

- Size - Breite und Höhe des Rasters.
- Rows - Anzahl der Reihen im Raster oder Rumpf.
- Column - Anzahl der Spalten im Raster oder Rumpf.

#### ***Line:***

Erstellt ein Polygon oder eine Linie.

- Primitive Type - Geometrie Typ
- Origin - Start der Linie

- Direction - Richtung der Linie
- Length - Länge der Linie
- Points - Anzahl der Punkte mit der die Linie erstellt wird

### **L-System:**

Das Lindenmeyer-System ist ein Algorithmus zum Umschreiben von Zeichenfolgen. Einfach gesagt ersetzt der Algorithmus nach einer bestimmten Regel Zeichen durch andere Zeichen bzw. Zeichenketten. Durch Verwendung von Iterationen, wobei die Ergebnisse zur Grundlage der nächsten Iteration werden, kann man ein Wachstum darstellen. In unserem Fall nutzen wir sie, um Blumen und deren Verzweigungen am Stängel herzustellen.

- GEOMETRY:
  - Type - Skeleton = einfache Linien; Tubes = Röhren
  - Generation - gibt an wie viele Generationen im Falle einer Iteration durchlaufen werden
  - Apply Colour - durch setzen werden die Farben der eingefügten Objekte übernommen
- TUBES:
  - rows & cols - gibt Anzahl der Reihen und Spalten des 'netztes' der Röhren an
  - Tension - wie gerade die Röhren zu ihrem Zielpunkt gehen
  - Thickness - breite der Röhre
- RULES:
  - Premise - Ausgangszustand des L-Systems - Generation 0
  - Rule# - Regeln/Formeln die den Verlauf des L-Systems beschreiben

Befehle die genutzt werden:

- [] - erstellt einen neuen Branch
- {} - erstellt ein Polygon
- ABCD - Variablen die für die einzelnen Regeln stehen
- JKM - verweist auf die Objekte bzw. die Eingänge
- F(a) - erstellt eine einfache Linie. Man kann sie zusätzlich durch einen Wert in einer Klammer verkürzen/verlängern
- . - Polygon Vertex (Vertex = Ecke eines Polygons)
- !(s) - Multipliziert momentane Breite mit s. In unserem Fall bewirkt es, dass die Röhren immer schmäler werden
- &(a) - nach oben kippen um a grad

- $\hat{a}$ ) - nach unten kippen um a grad
- "(s) - multipliziert bzw. skaliert die momentane Branch länge mit s
- /(a) - dreht im Uhrzeigersinn a grad
- -(a) - rotiert nach links um a grad
- +(a) - rotiert nach rechts um a grad

### ***Sphere:***

Sphere ist eine einfache Node, die ein Kugel Objekt erstellt. Man kann bei dem Primitive Type einstellen, welche Oberfläche die Kugel haben soll. Man kann den Radius bestimmen, die Frequenz und auch wie die Höhe und Breite der Kugel, dadurch kann man zum Beispiel auch Ovale formen.

### ***Tube:***

Ein Tube ist ein einfacher Kegel. Häufig genutzte Parameter:

- Radius - Radius der Tube.
- Radius Scale - Einheitlicher Maßstab für den Radius.
- Height - Die Höhe der Tube.

### ***Utility:***

#### ***Convert:***

Konvertiert eine Geometrie Art zu einer anderen Geometrie Art

- From Type - Welche Art die Geometrie hat
- Convert To- Welche Art die Geometrie danach haben soll

Division per Span

- Genaue Anzahl der Punkte
- U- Punktdichte in U Richtung
- V - Punktdichte in V Richtung
- U Order- Spline-Reihenfolge von Kurven und Flächen in U
- V Order - Spline-Reihenfolge von Kurven und Flächen in V
- Interpolate Through Hulls - Die Form der Geometrie wird übernommen, deaktiviert U & V

### ***Copy Stamp:***

Copy besitzt zwei Hauptaufgaben: - mehrere Kopien einer Geometrie erstellen - Kopiert die erste Geometrie an Punkte der Zweiten Geometrie Die einzelnen Kopien können zusätzlich mit COPY transformiert werden. Durch STAMP -> Stamp Inputs können Variablen für jede Kopie weitergeführt werden.

### ***Copy to Points:***

Wie der Name schon sagt, kopiert es ein Objekt (erster Input) an übergebene Punkte (zweiter Input). Wird in unserem Fall innerhalb einer Schleife genutzt

### ***Merge:***

Diese Node führt die Geometries aus seinen Eingaben zu einem einzigen Geometriestrom zusammen, den Sie dann durch andere Nodes senden können. Sie fügt quasi zwei Objekte zusammen. Die Merge-SOP wendet alle eingehenden Attribute auf die gesamte Eingabegeometrie an. Jede Eingabegeometrie kann ihren eigenen Satz von Attributen haben.

### ***Mirror:***

Dupliziert und spiegelt das Objekt.

### ***Null:***

Das Null Objekt dient zum einen als Platzhalter. Es kann verwendet werden, um einen Ort im Raum der Scene zu bestimmen oder als 'Betrachtungsobjekt' als Hilfe für die Koordination in der Szene. Seine Parameter ähneln denen von 'Transformieren' und 'Verschiedenes'. Es kann ein Objekt Transformieren und hilft später besser auf das Objekt referenzieren zu können

### ***For Each (begin & End):***

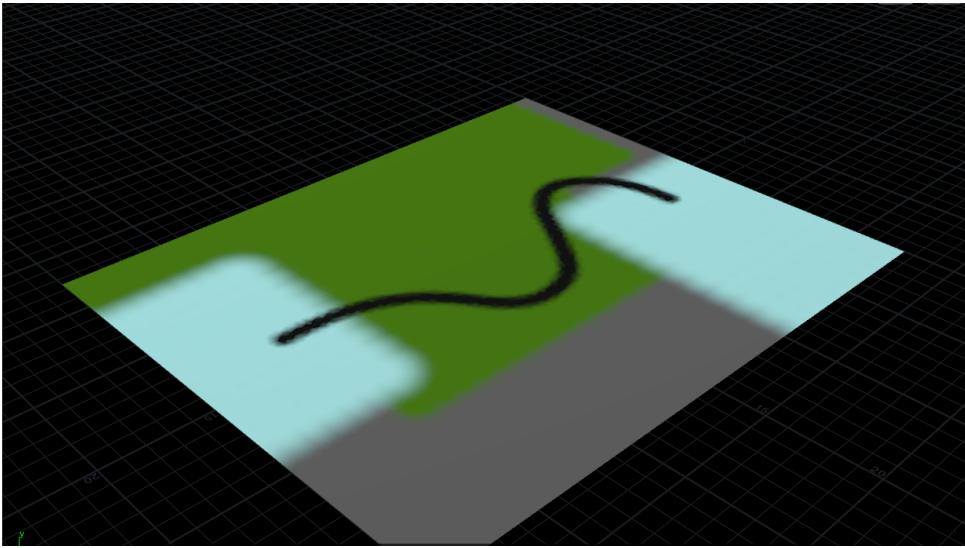
For Each beinhaltet zwei Nodes: begin und end. Der Context der Schleife steht in der Input-Node der begin-Node, was in unserem Fall unsere Punkte sind. Die For Each Schleife geht nun jeden Punkt durch und wendet die Nodes die zwischen begin und end liegen auf jeden dieser Punkte an.

## **Tutorials:**

### **Terrain:**

1. Um unser Terrain zu erzeugen benötigen wir als aller erstes einmal ein **Feld**. Mit der **Grid-Node** erzeugen wir ein (in diesem Fall 25 x 30) Feld. Die Anzahl der **Zeilen & Spalten** werden nach Gemütslage erhöht. Je höher die Anzahl, desto mehr Details (Primitives) existieren im Feld.

2. An unser Grid setzen wir ein Remesh, damit wir später noch einmal besser mit den Details im Feld arbeiten können.
3. Nach dem **Remesh** wird unser komplettes Gitter in einer Farbe eingefärbt. Die Farbe selbst ist prinzipiell egal, wichtig ist in unserem Setup der **Farbwert für Rot**. Wir benutzen hier einen Grauton, der die Basis der Szene stellt.  
Wir haben damit unsere Terrain-Basis gelegt. Im folgenden werden noch spezielle Teilgebiete angefertigt.
4. Neben dem Basisterrain-Setup liegen unsere Teilgebiet-Nodes. Wir erschaffen hier jeweils eine **Gruppe** für Grassland und die Berge, die später generiert werden sollen. Dazu werden beide Group-Nodes jeweils in unser erstes Grid eingespeist. Wir wählen die **Base Groups**, die wir gleich weg löschen wollen, per Knopfdruck aus und drücken Enter.
5. An unsere Gruppen hängen wir jeweils ein **Boolean** an. Wir setzen die Basis-Gruppe B (also Grassland/Berge) und *subtrahieren* A-B. Am Ende sollte nur unsere gewünschte Fläche übrigbleiben.
6. Diese Flächen werden anschließend wieder in Farben eingefärbt (nochmal: der **Rot-Wert** ist hier wichtig!)
7. Neben Grass, Gebirge und Grundland wollen wir noch einen Pfad in unserer Szene haben. Dazu wählen wir die **Add**-Node aus und beginnen Punkte in der Scene-View zu setzen.
8. Da unsere Punkte und damit unser Pfad für später grob gesetzt sind können wir jetzt die Linie zu einer feineren Kurve formen. Eine häufig genannte Methode ist die **Convert**-Node, bei der unsere Punkte zu einer *NURBS-Kurve* geformt werden. Wir verwenden hier jedoch **Subdivide**. Subdivide gibt uns mehr Punkte entlang unseres Pfades, da unsere Tiefe auf 3 gesetzt ist.
9. Egal wofür man sich entscheidet, daran wieder eine **Color**-Node anhängen und diesmal auf *schwarz* setzen.  
Damit steht jetzt unsere Karte! Jetzt wo wir die Aufteilungen für später definiert haben wollen wir unsere Teile zusammenfügen bzw. überlappen.
10. Hierzu wählen wir die Node **Attribute Transfer** dreimal hintereinander. Unsere Basis ist erst unser... Basisgrid! Der zweite Input ist dann unser Grassland. Attributtransfer transferiert diesen zweiten Input auf das Basisgrid und erschafft ein neues, verschmolzenes Grid. Selbes wird dann mit den Bergen und dem Pfad jeweils wiederholt. Am Ende sollte ein Feld mit 4 verschiedenen farblich unterlegten Teilgebieten zu sehen sein.

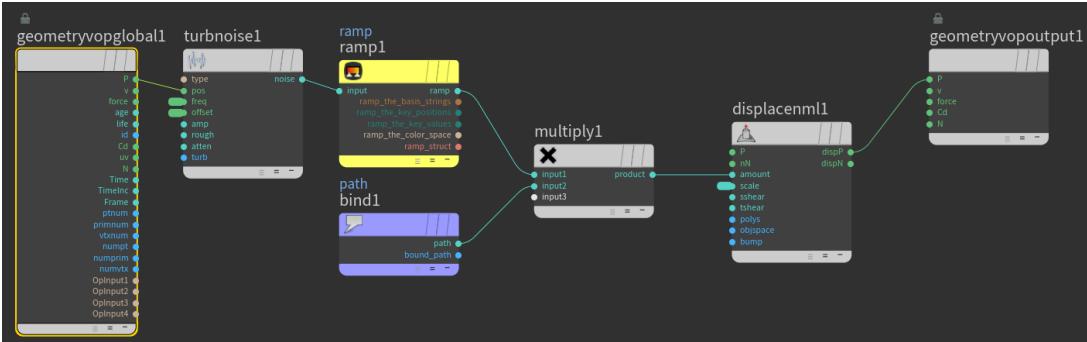


11. Wir fügen an dieser Stelle ein **Transform** ein, um unsere gesamte Szene leicht zu kippen. Das mag an anderer Stelle auch möglich sein, jedoch haben wir hier den Vorteil, dass alles was später an Generation erfolgt von dieser Schieflage ausgeht und auch wirklich nach oben zeigt!
12. Nach dem Transform schreiben wir mit **Attribute Wrangle** eine Expression:

`f@path = @Cd.r;`

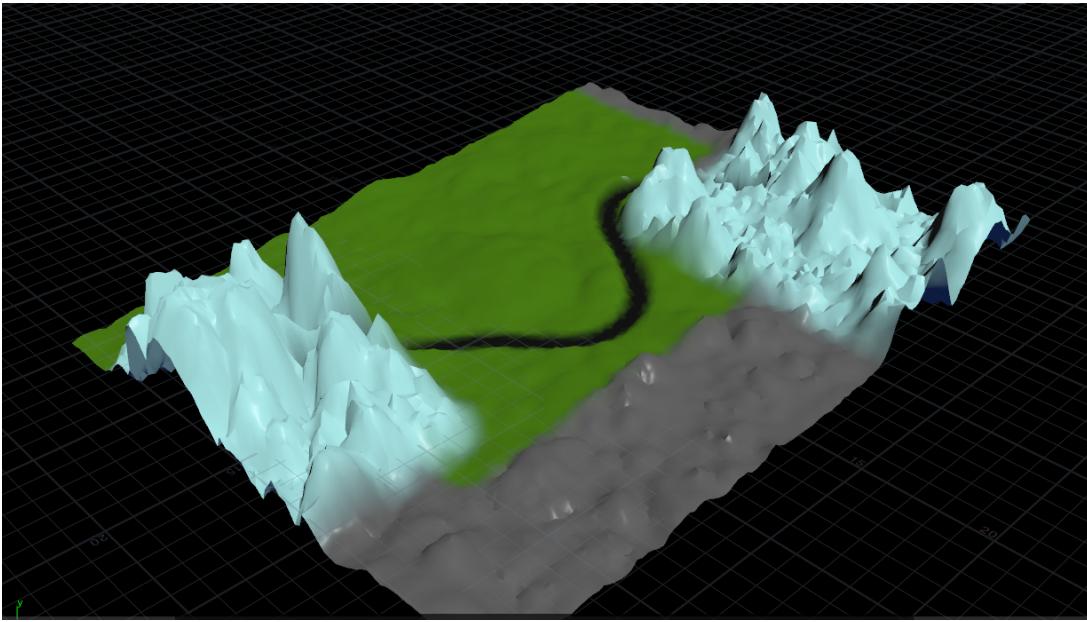
Wir erschaffen hier also eine Float-Variablen namens 'path', deren Wert mit dem Rot-Wert der einzelnen Teile belegt wird. Ergo hier bei uns alles was den Pfad darstellt bekommt den Wert 0, das Grassland den Wert 0.15, das Gebirge 1 und der Rest 0,3.

13. Hier wird dann anschließend das erste große vollbracht. In der **Attribute VOP**-Node sehen wir (nachdem wir in die Node reingehen) einen Haufen an Einstellungen:



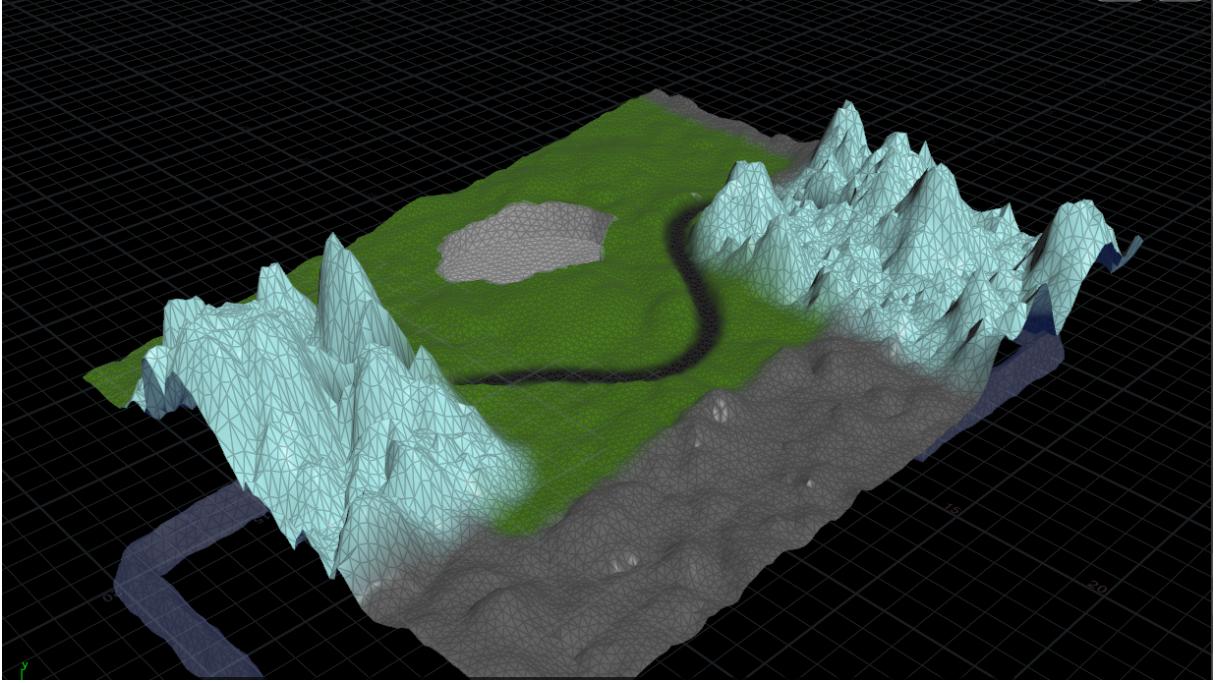
Hier wird in zwei Schritten entschieden wie viel Noise-Effekt auf unser Terrain projiziert wird. Es gibt einmal ein generelles Grundrauschen, dass der obere Pfad festlegt. Das untere **Bind** importiert unsere Path-Variablen und multipliziert den jeweiligen Wert mit unserem Grundrauschen. Sprich unser Grassland bekommt ein Grundrauschen \* 0.15, unser Gebirge das volle Grundrauschen und unser Pfad bleibt genauso wie er ist.

Unser Terrain steht jetzt!



14. Da wir Höhlen in unseren Bergen haben wollen bauen wir uns noch zwei Tunnel. Wir nehmen uns zwei **Tubes**, *schließen* die End Caps und finden Werte, die uns gefallen. Eine dreieckige Form passt zu unserer Szene später ziemlich gut, also ziehen wir die *Columns auf 3 runter*. Die *Height* legt die Länge unserer Röhre fest, da wir sie auf der X und Z Achse um 90° gedreht haben liegt diese auch flach auf. *Rows* können bei der **Bend**-Operation wichtig sein, dazu aber dann später mehr.
15. Unsere Tubes bekommen dann eine Farbe, die wir später haben wollen. Der Farbwert Rot spielt hier übrigens keine Rolle mehr, da wir von dem Displacement auf dem Terrain abgekoppelt sind und unsere beiden Tunnel Geometrien *später* mit dem Rest zusammenfügen.
16. Mit **Polybevel** runden wir unsere Tube leicht ab.
17. **Bend** erlaubt es uns unsere Tubes in gewünschter Richtung um eine festgelegte Achse zu biegen, in unserem Fall um 90°.
18. Mit **Transform** passen wir die Position auf unsere jew. gewünschte Lage an.
19. Anschließend **Remeshen** wir unsere Tubes um für die nächste Node
20. **Mountain** mehr Dreiecke zum Verformen zur Verfügung zu stellen.
21. Da wir beide Röhren mit der Szene in einem Schritt erschaffen wollen, **Mergen** wir diese erst um dann anschließend in einer
22. **Boolean**-Node die beiden Inputs Terrain und Höhlen zu vereinigen, bzw. die Schnittmengen zu entfernen. Wichtig ist hierbei, dass die Eingänge der Höhle auch direkt an der Bergen anliegen!
23. Dieses neue Terrain bekommt noch ein kleines Becken für unseren See, den wir im Schnelldurchlauf folgendermaßen aufbauen: Tube > Transform > Mountain > Remesh

24. **Booleaniere** wieder beide Teile für ein neues Gesamtmesh und die Szene sollte so aussehen:

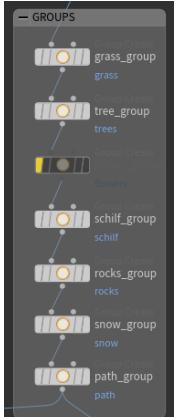


Für die nächsten paar Schritte benötigen wir wieder neue **Gruppen**. Zwar haben wir ja bereits ganz am Anfang schon welche definiert, Houdini kann mit diesen allerdings nicht mehr arbeiten, da diese Attribute auf unser Grid wieder und wieder transferiert wurden. Also legen wir einfach neue, etwas spezifischere Gruppen an.

Die Gruppen, die für uns relevant sind, sind:

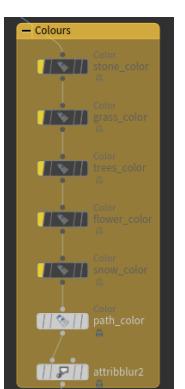
- Gras
  - Baum
  - (Blumen)
  - Schilf
- 25.
- Steine
  - Schnee
  - Pfad

Diese werden alle per Group Select in der Scene View ausgewählt, um später Scatter-Points und Farben drauf setzen zu können.



- Nach unseren Groups wandern wir hier in zwei Richtungen: einmal links ist unser gesamt Scatter-Point-Setup, welches uns erlaubt Objekte in der Szene einzuspeisen. Rechts färben wir unsere Szene mit den gewünschten *eigentlichen Farben* ein. Da unsere Anfangsfarben soweit aber zur Szene passen sind diese jedoch alle bis auf die Weg Farbe ausgegraut.
- 26.

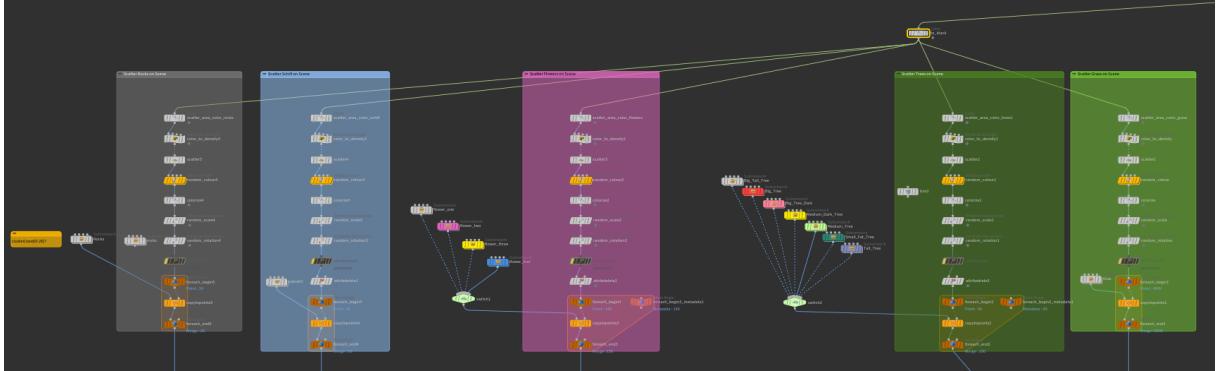
Die Farbe des Pfades ändern wir, indem wir bei der Color-Node die Gruppe anwählen.



27. **Attribute Blur** verwischt die Grenzen unseres gewählten Attributes (*Cd* für Color)

in der gesamten Szene.

28. Nach dem Aufbau unserer Grund-Terrains und dem Setzen der Farben reduzieren wir noch die Polygon-Anzahl mithilfe von **PolyReduce** auf 50
29. **Edgecusp** (bzw. **Facet**) daran gekettet sorgen für schöne Lichteffekte in unserem Low-Poly-Setup, da die Flächen härtere Übergänge bekommen.  
Hieran angekettet wird nur noch ein letztes **Merge**, deshalb springen wir noch einmal zurück zur eben genannten Gabelung.



Dieses Setup ist ziemlich groß, jedoch passiert hier in jedem Block prinzipiell das-selbe.

30. Wir fangen mit einer weiteren **Color**-Node an. Diese färbt unsere gesamte Szene in schwarz ein, um in den nächsten Schritten einen möglich hohen Kontrast zu ermöglichen. Nehmen wir als Beispiel nun das Gras-Setup ganz rechts:
31. Unser erster Knoten ist wieder eine **Color**-Node. Hier wird eine unserer vordefinierten Gruppen ausgewählt (in diesem Fall „grass“) und die Farbe weiß auf eben nur diese Fläche, die von *grass* beansprucht wird, verteilt.
32. Daran angeknüpft schreiben wir in **Attribute Wrangle** eine neue Float-Variablen *density*. *Density* merkt sich wiederum die Rot-Werte aus dem ihr gegebenem Input – sprich alles was weiß markiert ist.
33. In **Scatter** importieren wir unser *density*-Attribut und tun damit unsere Punkte deutlich dichter als gewöhnlich setzen. In unserem Gras-Block haben wir hier 5000 Punkte gesetzt, also wird später 5000-mal je ein Gras-Objekt auf einen Punkt gesetzt.
34. (In der **Attribute VOP** setzen wir zufällige Farben für unser Gras. Hierbei wird jedem einzelnen Punkt ein Wert zwischen 0 und 1 zugewiesen, der dann die Helligkeit bestimmt)
35. **Colorize** färbt dieses Farbattribut Cd dann mit einer von uns definierten Farbrampe ein.
36. **Attribute Randomize** nutzen wir, um diverse Attribute pro Punkt zufällig aufzusetzen. Hier bekommt jeder Punkt einmal ein pscale- und ein orient-Attribut, festgelegt durch ein Minimum und Maximum ausgehend vom Originalobjekt. Bei Pscale interessiert uns nur die Höhe, wir gehen also nur in eine Dimension. Orient ist für die Rotation verantwortlich, wir benötigen hier also 4 Dimensionen, wobei die nach oben [und unten] auf 0 gesetzt sind.

37. In einem **For-Each-Block** wird dann unser Gras eingespeist und auf jeden einzelnen Punkt (samt vordefinierter Attribute) gesetzt.

Bei den Blumen und Bäumen kommt hier noch ein **Switch** hinzu, das durch eine Funktion zufällig aus einer Inputmenge pro Punkt das nächste Objekt auswählt.

38. Unsere letzte Node im Terrainsetup ist ein finales **Merge**, dass das Terrain mit den dazu gecasteten Teilobjekten vereint!

### Bonus

Houdini erlaubt es uns auch Parameter in Abhängigkeiten zu anderen zu setzen. Was wir also beispielsweise machen können, ist eine **Null-Node** aufzusetzen, deren Aufgabe es ist, einige Kernparameter leicht erreichbar anzupassen.

Hier können wir also die Höhe der Szene und die Anzahl der Scatter-Punkte für jedes einzelne Teilobjekt festlegen!

Scene Height	<b>23</b>
Grass Scatter Count	<b>5000</b>
Tree Scatter Count	<b>100</b>
Flower Scatter Count	<b>150</b>
Schilf Scatter Count	<b>50</b>
Rock Scatter Count	<b>25</b>

### KAMERA-Drohne

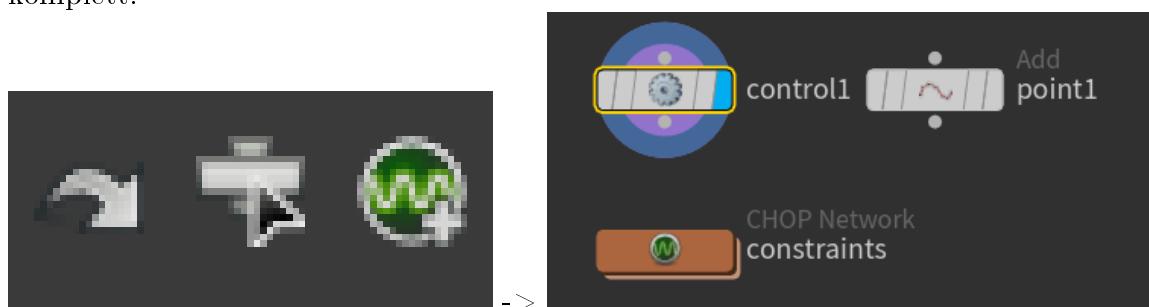
Die Kamerafahrt ist vom Aufbau relativ simpel. Was wir brauchen sind:

- einen **Pfad**, den die Kamera entlangläuft
- eine **Null-Node**, die die Position der Kamera relativ zur Zeit trackt
- eine **Kamera** (duh)

Der Aufbau des Pfads läuft genauso ab wie der innerhalb des eigentlichen Terrains. Erzeuge in einem neuen Objekt *cam\_path* Punkte und schaffe eine Kurve aus diesen. Ursprünglich war gedacht, dass der Kamerafahrt exakt derselbe Pfad ist, bloß ein wenig nach oben versetzt. Wir haben jedoch einen interessanteren Weg für diese Fahrt gefunden und damit dies Punkte nochmal manuell von Hand angepasst. Neu hinzu kommt hier eine *CAM\_SUB\_OUT* benannte **Null-Node**.

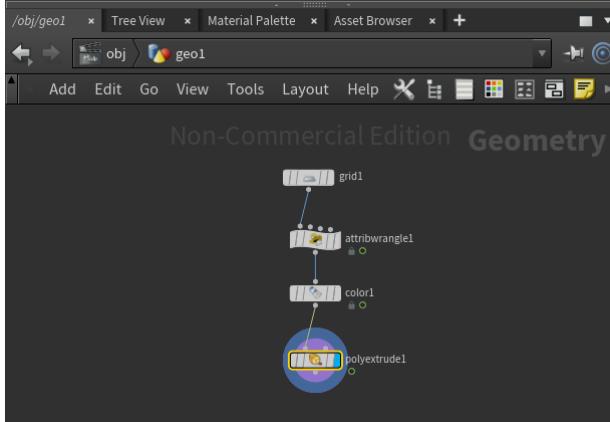
Zurück in der Objektansicht sehen wir noch eine weitere **Null-Node** mit dem Namen *PATH\_SUBDIVIDED*. Diese wird in die **Kamera** eingespeist und sagt dieser, wie die wo langzufahren hat.

Beim Erzeugen dieser Null setzen wir den Haken bei *Enable Constraints* auf ja und sehen ein kleines Feld aufzploppen. Ganz rechts haben wir die Möglichkeit ein Constraint direkt zu erschaffen, alternativ machen wir das manuell in der Node selbst. Außerdem wollen wir hier einen Slider für die Kameraposition hinzufügen, diese Null steuert also die Kamera komplett.



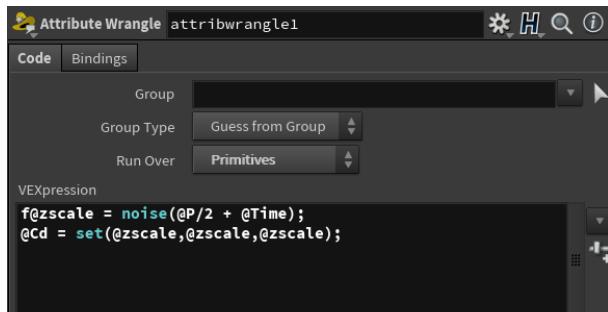
Klicken wir uns nun in das Constraints-CHOP Network rein sehen wir die **getworld-space**-Node. An diese hängen wir ein **Follow Path** an und setzen den SOP Path auf unseren Kamerapfad *CAM\_SUB\_OUT* und schreiben unter *Position* die Funktion `ch('..../pos2')`, wobei pos2 der vorher hinzugefügte float-Wert 'Position' der Null-Node ist. In PATH setzen wir dann die Position 0 auf unser Startframe und 1 auf unser Zielframe. Den Rest dazwischen berechnet Houdini selbst.

## Wasser:



Zuerst wird eine Geometry erstellt. In dieser Geometry wird eine Grid-, eine Attribute Wrangle-, eine Color- und eine Poly Extrude-Node benutzt.

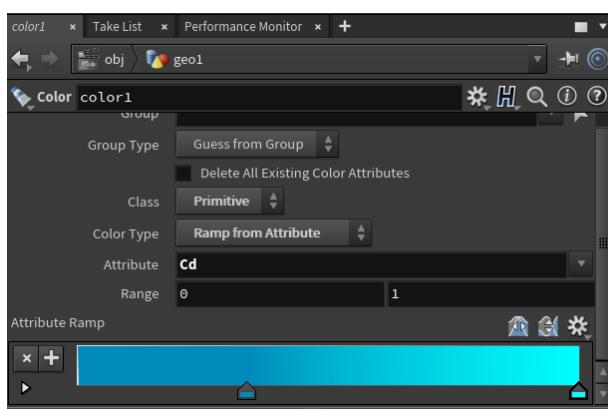
- Das Grid wird zuerst gesetzt und es können die Werte von rows und Columns (je mehr desto 'realistischer'), sowie Länge und Breite geändert werden.



Als Zweites kann man die Attribute Wrangle-Node setzen und definiert die Attribute mit einer 'VEX expression':

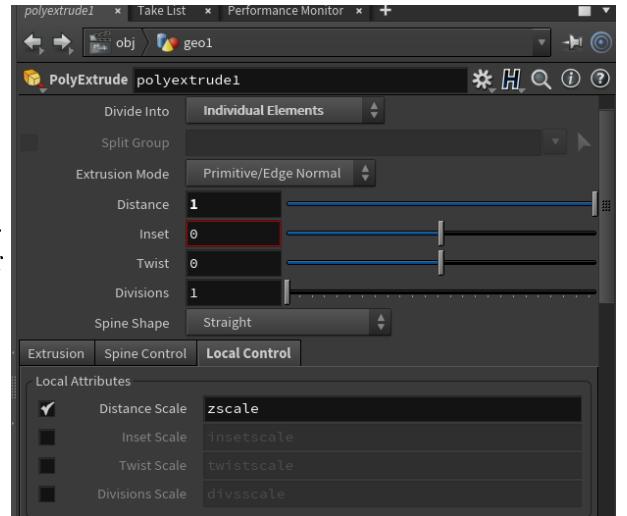
Wir benutzen dies hier, um an manchen Stellen auf dem Grid dunklere und hellere Flächen in Abhängigkeit von der Zeit zu setzen, die die Wellen des Wassers darstellen.

2.

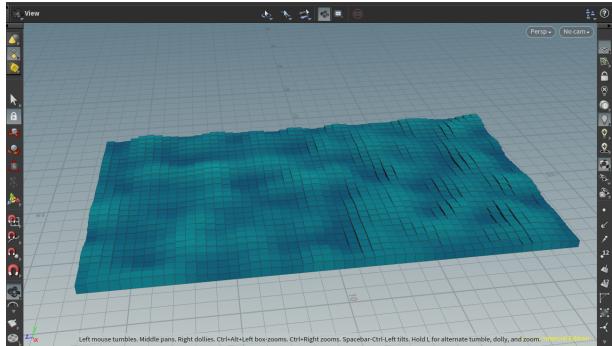


Mit Color kann man die gewünschte Farbe angeben, die in diesem Fall Blau ist.

3.



- Als letztes setzt man die Poly Extrude-Node.  
 4. In dieser Node werden die Werte folgender Maße gesetzt:



Das Objekt sollte damit so aussehen:

## Steine:

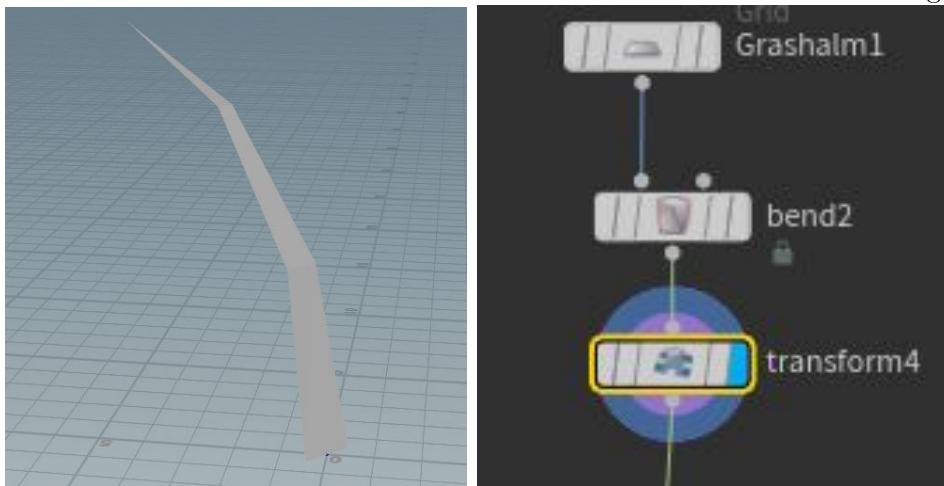
Die Steine bestehen aus einem sehr simplen Aufbau von nur vier Nodes. Eine normale **Sphere**, die etwas *gestaucht* wird um die Steine etwas flacher zu gestalten. Diese wird mit einer '**Mountain**' Node uneben gemacht und mit einer '**Normal**' Node die Kanten eckiger gemacht, um den Low Poly Stil umzusetzen. Anschließend wird die Sphere mit einer '**Color**' Node grau eingefärbt.

## Gras:

Um Gras zu modellieren gibt es, wie bei allem, viele verschiedene Möglichkeiten. Die einfachste davon ist es das **Hair-Tool** zu benutzen. Das Hair Tool ist ein eingebautes Houdini-Tool um eigentlich *Fell*, oder *Haare* zu generieren. Man kann es jedoch auch benutzen um flächendeckend realistisches Gras zu bauen. Das Hair Tool bietet neben der Einfachheit auch den Vorteil, dass die Berechnungen der Polygone von Houdini automatisiert und optimiert sind. Dadurch wird dem Problem der Performance-, was beim händischen modellieren von realistischem Gras auftritt, vorgebeugt, da man sehr schnell eine sehr hohe Anzahl an Polygonen erreicht. Da wir uns in unserem Projekt jedoch für eine Low-Poly Lösung entschieden haben, werde ich darauf nicht genauer eingehen, empfehle jedoch das Tutorial von CGI Nerd. (<https://youtu.be/7bRr38S59i4>) Für unser Projekt habe ich mich entschieden, das Gras händisch zu bauen:

1. Als erstes wird nur ein Grashalm modelliert. Dazu wird ein **Grid** mit 2 *Rows* be-

nutzt. Ich habe mich für ein Größenverhältnis von 10(Länge)x1(Breite) entschieden. Außerdem habe ich die Anzahl der *Columns* auf 4 gesetzt, um ein eckiges Ergebnis zu erzielen, je mehr Columns hier gesetzt werden, desto runder wird später die Biegung des Grashalms. Als nächstes wird das Grid mit der **bend**-Node gebogen. Die Stärke der Biegung wird nicht als fester Wert fest gelegt, sondern erhält einen *zufälligen Wert*. Dies wird erreicht, indem man den Wert der in der später benutzten **Copy Stamp**-Node referenziert. Außerdem wird das *Taper*-Attribut gesetzt, um das Grid spitz zulaufen zu lassen. Als letztes wird der Grashalm mit der **Transform**-Node neu ausgerichtet. Auf jeden Fall muss er um 90 Grad in z Richtung gedreht werden, damit die Spitze nach oben zeigt. Um etwas Abwechslung in das Gras zu bringen, wird sowohl die Rotation auf der x-Achse als auch die Größe zufällig gesetzt.



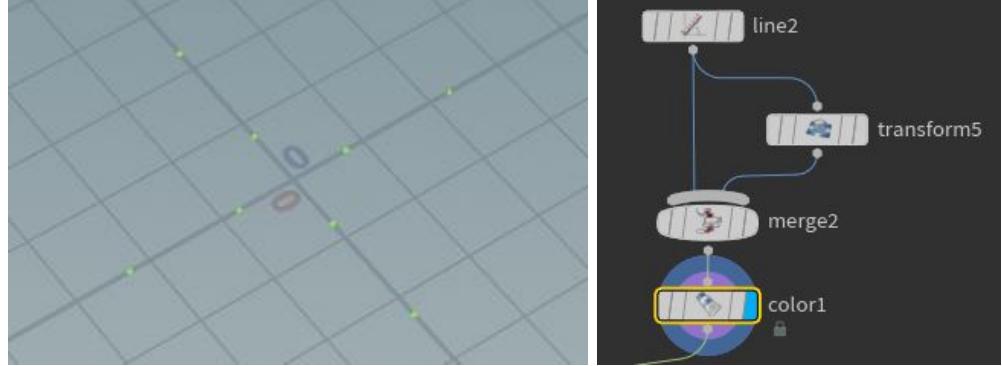
2. Als nächstes wird der Grashalm zu einem Grasbüschel zusammengesetzt. Dazu habe ich zwei verschiedene Dinge ausprobiert:

- Erster Ansatz: Ein normaler **Circle** als Grundfläche der um -90 Grad auf der x-Achse gedreht wird, damit er auf dem Boden liegt. Auf der Fläche werden dann mit **Scatter** so viele Punkte verteilt, wie man Grashalme haben möchte. Ich habe mich für 8 entschieden.
- Zweiter Ansatz: Um die Anzahl an Grashalmen zu reduzieren und trotzdem den Eindruck von flächendeckendem Gras zu erwecken, bedienen sich viele Spiele dem Trick 2D Grashalme in einem Kreuz anzuordnen.

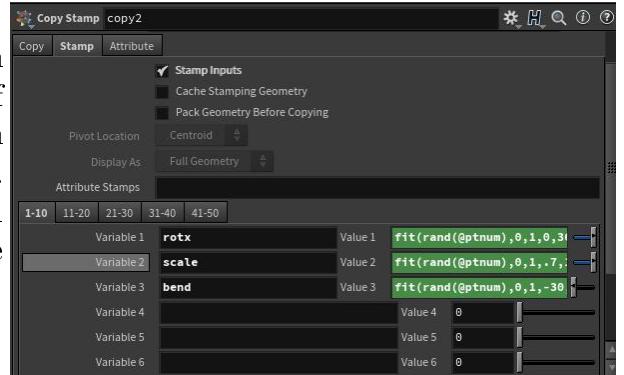


Dies hab ich versucht nach zu bauen, um zu schauen welche Version besser für unsere Zwecke geeignet ist. Dazu habe ich eine Linie **Linie** aus *Points* erstellt: die *Länge* kann beliebig gewählt werden und die Anzahl der Points ent-

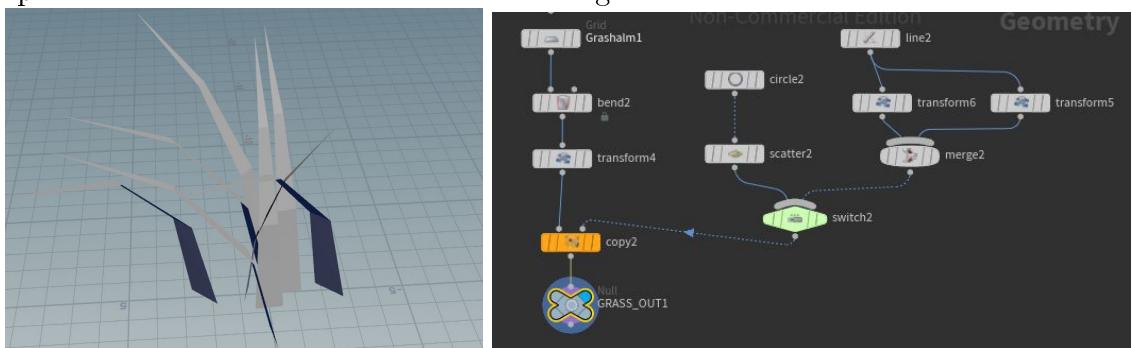
spricht der halben Anzahl der Grashalme. Anschließend hab ich die Linie mit einer **Transform**-Node um 90 Grad um die y-Achse gedreht, und habe die transformierte- mit der ursprünglichen- Linie durch eine **MergeMerge**-Node zusammengefügt. Dadurch erhält mein ein Kreuz aus Punkten. Um die Punkte im Screenshot besser sichtbar zu machen, habe ich sie mit einer **Color**-Node grün eingefärbt.



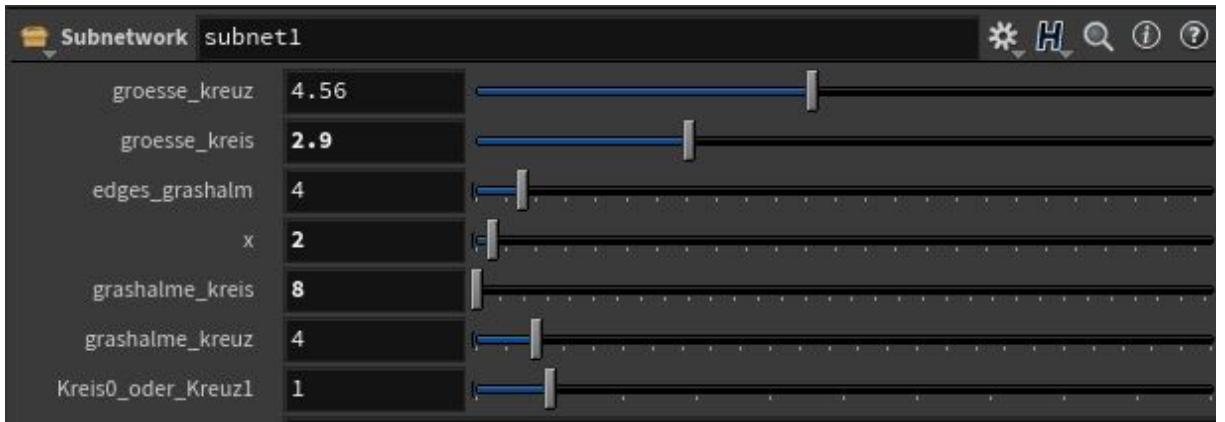
- Als letztes wird der Grashalm mit einer **Copy Stamp**-Node auf jeden Punkt von entweder dem Kreis, oder dem Kreuz gesetzt.
- Als Stamp-Input werden die drei Variablen für die Rotation, die Größe und die Biegung gesetzt.



Mit der **Switch Node** kann nun zwischen den beiden Versionen frei gewählt werden. Die Variable kann in den Parametern der vorher genannten Nodes über den angegebenen Namen referenziert werden. Durch das rand(@ptnum) wird eine zufällige Zahl für jeden Punkt generiert. Durch fit werden die Werte einem angegebenen Zahlenbereich zugeordnet. fit(n, oldmin, oldmax, newmin, newmax) wird verwendet um eine beliebige Zahl, in unserem Fall die Nummer des Punktes, aus einem Zahlenbereich einem anderen Bereich zuzuordnen. Am Schluss hängen wir eine **Null-Node** an um später von außen einfacher auf das Gras zugreifen zu können.



In dem wir alle Nodes in einem **Subnet** speichern, können wir von dort aus auf ausgewählte Parameter zugreifen und diese übersichtlich anpassen.



## Schilfrohr:

Das Schilfrohr besteht aus drei Teilen: Die Blüte, der Stiel und eine Fläche um wie bei dem Gras mehrere Stiele zusammen zu setzen.

### Blüte:

Die Blüte besteht aus einer **Polygon Tube**, der *Radius* und die *Höhe* kann beliebig gesetzt werden, sollte aber im Verhältnis zum Stiel stehen. Ich habe mich für einen Radius von 0.037 und eine Höhe von 0.27 entschieden. Die *End-Caps* sind an. Damit die Blüte rund wird, habe ich die *Columns* relativ hoch, auf 12, gestellt.

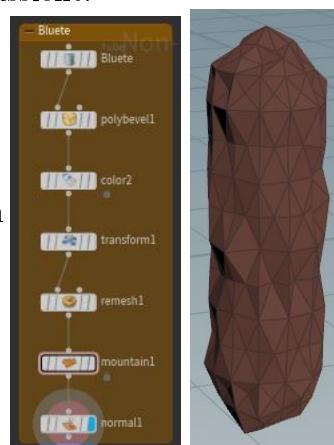
Anschließend werden die End Caps mit der **Polybevel**-Node abgerundet. Ich habe die *Distanz* auf 0.044 und die *Divisions* auf 3 gesetzt. So erhalten wir eine TikTak ähnliche Form.

Diese wird mit der **Color**-Node braun gefärbt und anschließend mit einer **Transform**-Node auf der y-Achse so weit nach oben geschoben, bis sie auf den Stiel passt. Ich fand den Wert 0.34 für mein Beispiel passend.

Anschließend verwende ich die **remesh**-Node, um danach mit der **Mountain**-Node die Blüte etwas deformieren zu können. Ich wollte keine große Veränderung, sondern nur dass die Blüte etwas uneben wird. Daher habe ich die Werte mit einer *height* von 0.02 und einer *Element Size* von 0.0001 sehr niedrig gesetzt.

Um dem Low Poly Stil gerecht zu werden, verwende ich zum Schluss eine **normal**-Node, welche dafür sorgt, dass die Blüte eckiger aussieht.

Das Network und die Blüte sehen dann in etwa so aus:

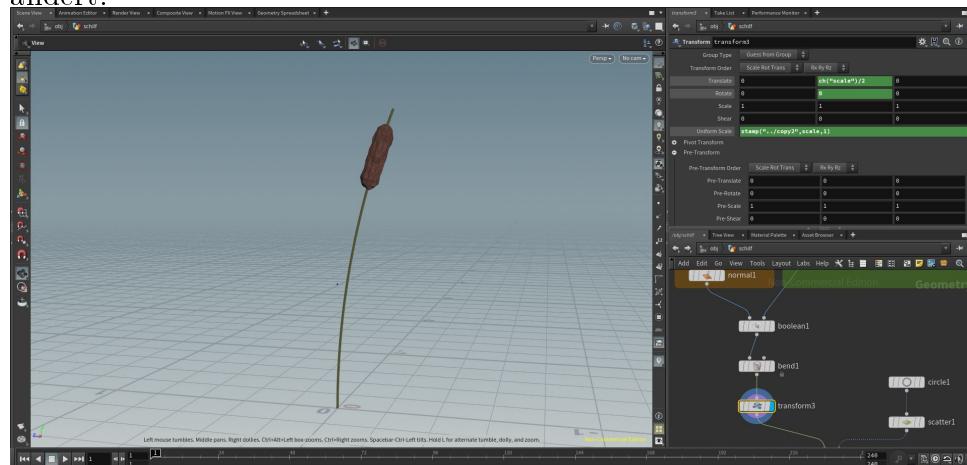


## Stiel:

Der Aufbau des Stiels ist ähnlich dem der Blüte jedoch viel einfacher, er besteht nur aus einer **Tube**. Die *Höhe* habe ich auf 1 gesetzt und die *Columns* gerade so hoch, dass der Stiel rund wird. Die Anzahl der *Rows* habe ich relativ hoch, auf 41, gesetzt, damit später beim Biegen des Stiels gute Ergebnisse erzielt werden können. Anschließend habe ich den Stiel mit einer **Color**-Node grün gefärbt.

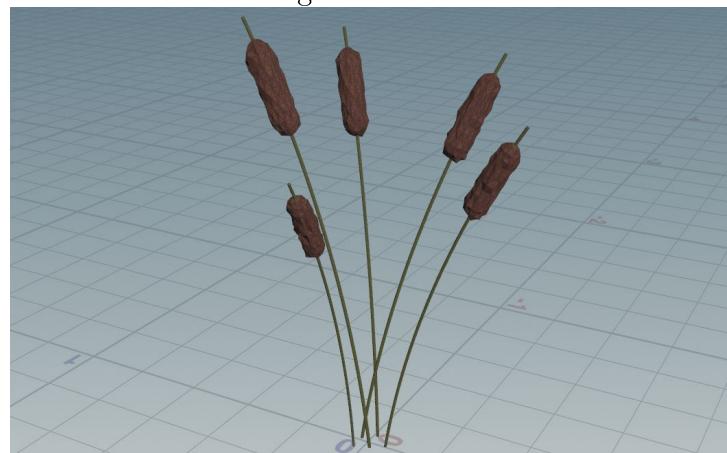
## Zusammenfügen:

Als erstes wird die Blüte und der Stiel mit Hilfe einer **boolean**-Node zu einem Objekt zusammengefügt, welches dann mit **Bend** gebogen werden kann. Der Winkel der Biegung wird, wie bei dem Gras, durch das Referenzieren eines Attributes in der Copy Stamp Node zufällig gesetzt. Das gleiche geschieht mit Hilfe einer **Transform**-Node für die Größe der Stiele. Außerdem wird mit der Transform Node die Pflanze nach oben verschoben, bis sie auf dem Boden steht. Dies passiert prozedural, indem der *y-translate*-Wert die Höhe des Stiels referenziert. So wird die Pflanze immer genau um die Hälfte der Höhe nach oben verschoben, und ist immer am Boden ausgerichtet, auch wenn sich die Höhe des Stiels ändert.



Anschließend werden die Stiele, genau wie beim Gras, durch eine **Copy Stamp**-Node mit Zufallswerten auf Punkte übertragen. Diese wurden wieder mit Hilfe der **Scatter**-Node aus einem Kreis extrahiert. Man kann mit dem *Offset* der Scatter Node herumspielen, bis man ein Ergebnis findet, das einem gefällt.

Dies war mein Endergebnis:

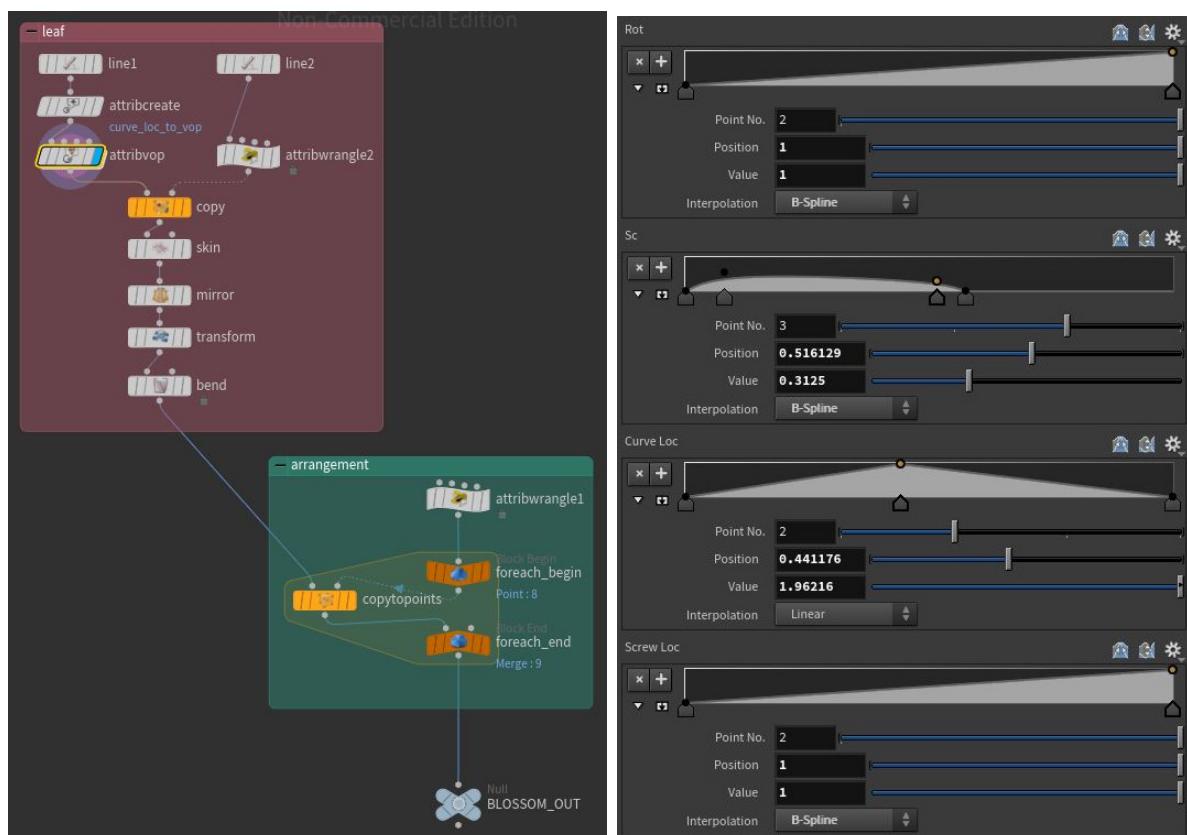


## Blumen:

### Blüte:

Um eine einfache Blüte zu machen, brauchen wir nur eins: ein Blatt. Um ein Blatt zu erstellen, brauchen wir erstmal zwei Linien. Line2(Direktion (0;0;1); Length 0.9; Points 8) bildet die Hauptader des Blattes. Ihr fügen wir die Attribute zum Verformen des Blattes wie z.b. die Rotation, Skalierung usw. mit Hilfe der Node Attribwrangle hinzu.

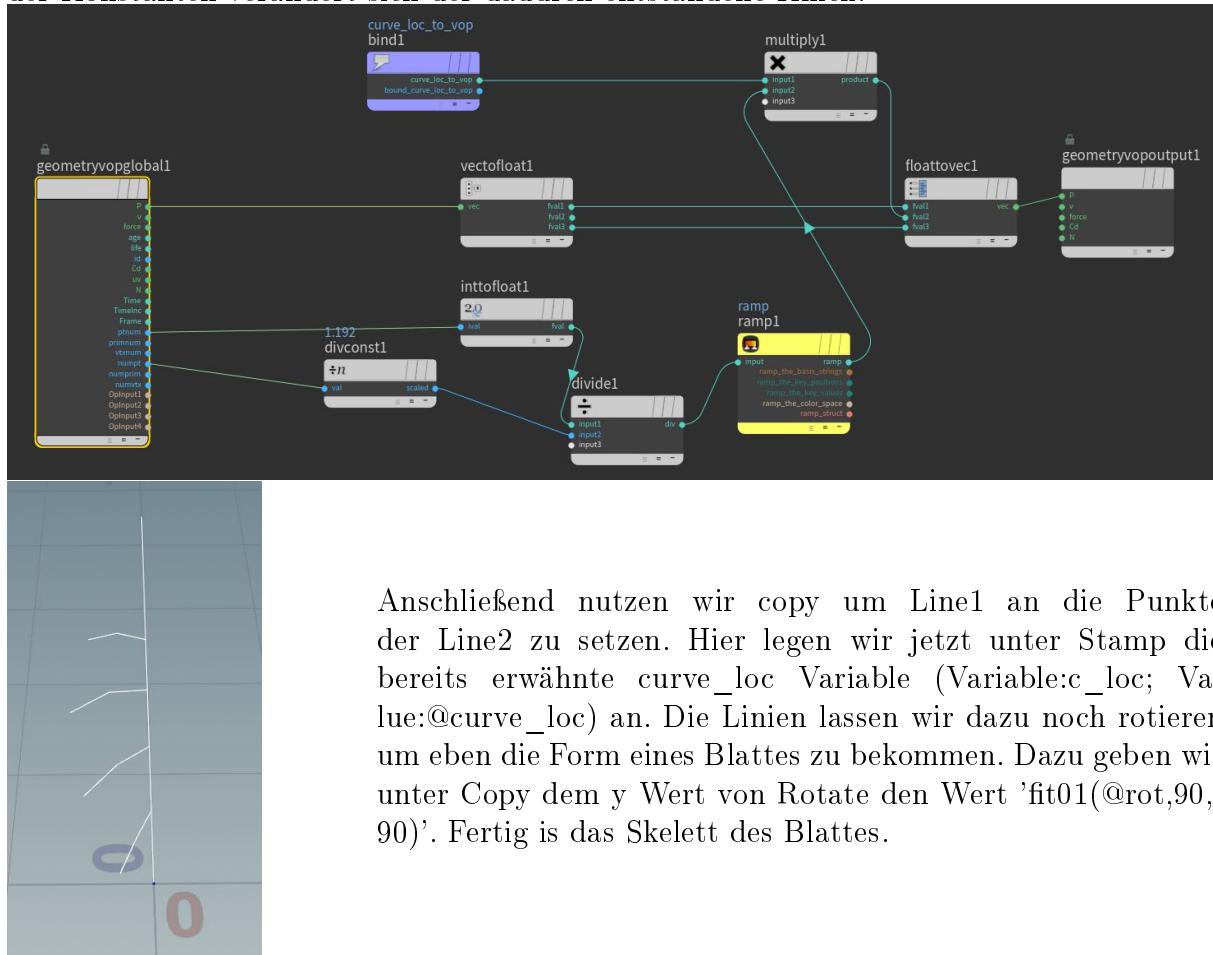
- $f@np = @ptnum / (@numpt * 1.0)$ ; -> np = aktuelle punkt / (anzahl pkt \* 1.0)
- $f@rot = \text{chramp}('rot', @np)$ ; -> erstellt ramp zur rotation (später bei Copy)
- $f@sc = \text{chramp}('sc', @np)$ ; -> erstellt ramp zur skalierung
- $@pscale = @sc$ ; -> setzt skalierung = sc
- $f@curve\_loc = \text{chramp}('curve\_loc', @np)$ ; -> curve loc ramp (später bei Line1 und Copy)
- $f@screw\_loc = \text{chramp}('screw\_loc', @np)$ ; -> screw loc ramp



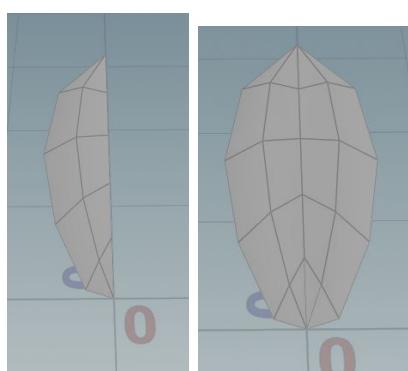
Durch sie können wir das Blatt nachher nach unseren Bedürfnissen anpassen und verformen, wodurch sich viele verschiedene Blüten bilden lassen würden. In unserem Fall bleiben wir bei einem Design, da unser Stil LowPoly ist. Da es bei den Linien schwer ist zu erkennen wie das Blatt im Nachhinein aussehen wird, werden wir die Attribute erst später genauer setzen.

Line1(Direction(1;0;0); Length 0.46; Points 3) stellen die Nebenadern dar. Ihr geben wir mit Attribcreate ein weites Attribut namens curve\_loc\_to\_vop, dessen Betrag wir der später folgenden Copy Node entnehmen und daher erst später setzen. Die Linie hat später einen Knick, wodurch sich eine Delle im Blatt bildet.

Dieses Attribut bearbeiten wir mit AttribVOP um die Werte der Position des Knicks anzupassen. Dazu nehmen wir die Position des Punktes und lassen x und z unberührt. y berechnen wir neu. Dazu nehmen wir den aktuellen punkt (ptnum), wandeln ihn zu einem float um (inttofloat) und teilen (divide) ihn durch die Anzahl der Punkte (numpt), welche durch (divconst) 1.192 geteilt wurde. Das Ergebnis geht in eine Ramp (Ramp) und wird anschließend mit dem Attribut curve\_loc\_to\_cop (bind) multipliziert. Durch verändern der Konstanten verändert sich der dadurch entstandene Knick.



Anschließend nutzen wir copy um Line1 an die Punkte der Line2 zu setzen. Hier legen wir jetzt unter Stamp die bereits erwähnte curve\_loc Variable (Variable:c\_loc; Value:@curve\_loc) an. Die Linien lassen wir dazu noch rotieren um eben die Form eines Blattes zu bekommen. Dazu geben wir unter Copy dem y Wert von Rotate den Wert 'fit01(@rot,90,-90)'. Fertig ist das Skelett des Blattes.



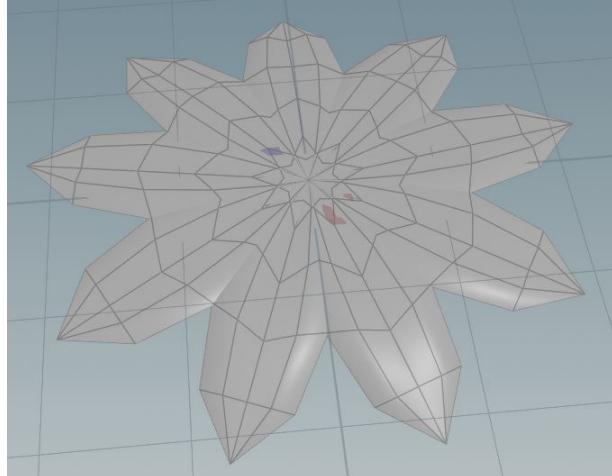
Um es jetzt zu einem Blatt zu machen, legen wir eine Haut (skin) darüber und spiegeln (mirror) es. Jetzt können wir zurück zu attribVOP gehe und die Ramps nach Belieben ändern. Optional kann das Blatt noch etwas gebogen (bend (Bend -53.8)) werden.

Jetzt muss das Blatt nur noch in einem Kreis positioniert werden und fertig ist die Blüte. Dazu geben wir dem Blatt noch eine Transform Node mit, in der wir bei Rotate y den Wert 'point(../foreach\_begin', 0, 'roff', 0)' geben. Auf den greifen wir gleich innerhalb der Schleife zu, um die Blätter mit zu rotieren. Als erstes erstellen wir eine For Each Schleife. In begin kommt ein attribwrangle.

```
int pnt = addpoint(0, {0,0,0});
float roff = chf('Angle') * @elemnum;
setpointattrib(0, 'roff', pnt, roff, 'set');
```

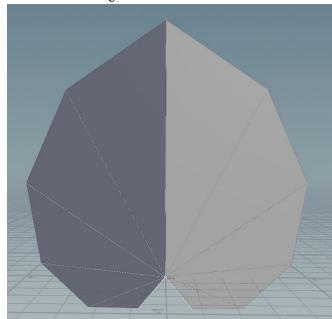
Wir legen also Punkte an, die alle bei 0,0,0 liegen an die die Blätter gesetzt werden, aber lassen das Blatt bei jedem Punkt rotieren (roff). Oberhalb gibt Number count (9) an, wie viele Blütenblätter erstellt werden. Unterhalb kann man mit Angle (160) den Winkel anpassen. Wenn man mehr oder weniger Blüten Blätter haben will, muss man den Winkel immer wieder neu anpassen.

Zwischen Begin und End kommt nun ein copy to points, dessen erster Input unser Blatt, zweiter Input Begin und output End ist. Dadurch wird an jeden Punkt ein Blütenblatt kopiert und dessen Rotation gesetzt. Zum Abschluss setzen wir eine Null Node die wir BLOSSOM\_OUT nennen, um später einfacher auf die Blüte zugreifen zu können. Fertig ist unsere Blüte.



### **Blatt:**

Das Blatt könnte man jetzt auf dieselbe weise wie das Blüten Blatt machen, da die Blätter am Stängel aber weniger wichtiger sind, gestalten wir dies etwas einfacher. Dazu nutzen wir L-Systeme.



```
Premise [A][B]
Rule 1 A=[+A{.}.C.]
Rule 2 B=[-B{.}.C.]
Rule 3 C=FFFC
```

Dazu erstellen wir Polygone (geometrische Figuren). Später Innerhalb der Blume verformen wir das Blatt, um es natürlicher aussehen zu lassen.

Am Ende erstellen wir wieder eine Null Node LEAF\_OUT.

## Kern:

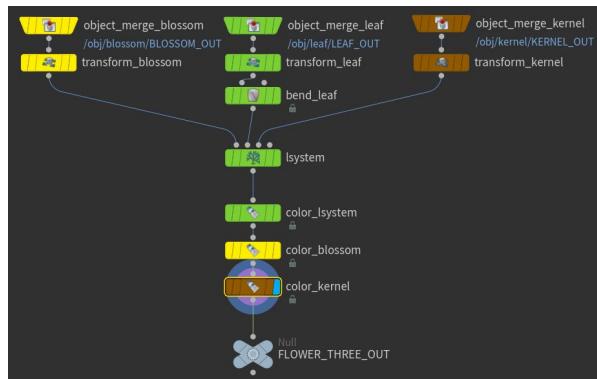
Da wir uns ja im LowPoly befinden, habe ich anstatt der größeren Anzahl an Kernen nur eine Sphere genommen, diese etwas verkleinert (Uniform Scale 0.2), gestaucht (Radius(0;0.4;0)) und später innerhalb der Blume braun gefärbt. Den Schluss bildet wieder eine Null Node KERNEL\_OUT.

Wenn man aber die Blume detaillierter darstellen möchte, sollte man auf die Anordnung der Kerner, welche sich Phyllotaxis nennt, achten. Diese bezieht sich auch auf die Blüten Blätter.

```
int num = chf("num");
float phi = (1 * sqrt(5) / 2.0;
float ang = 2* \$PI * (phi - 1) / phi;

for( int i=0; i<num; i++)
{
    float rad = i;
    vector pos =
        set(cos(ang*i)*rad ,0 ,sin (ang*i)*rad ;
    int pt = addpoint(0 , pos);
}
```

## Blume:

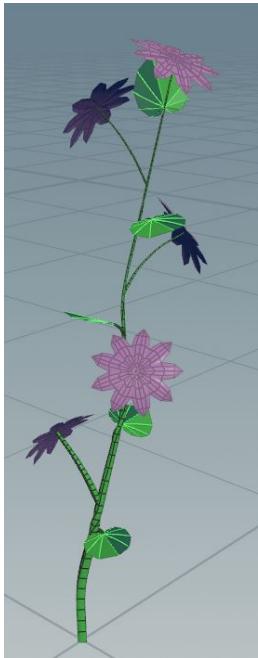


Jetzt brauchen wir nur noch alles zusammenzufügen und unsere L-Systeme zu erstellen. Um auf unsere Objekte zuzugreifen nutzen wir Object Merge. Dazu geben wir ihm unter Object1 jeweils eine Null Node. Jedem Object Merge hab ich jeweils noch ein Transform angehängt, um die einzelne Objekte anpassen zu können. Dem Blatt hab ich zusätzlich noch ein bend(Bend 22.5; Twist 55) gegeben, um es natürlicher wirken zu lassen. Die Objekte bilden den Input für unsere L Systeme (J = Blüte; K = Blatt; M = Kern)



Listing 1: Flower One

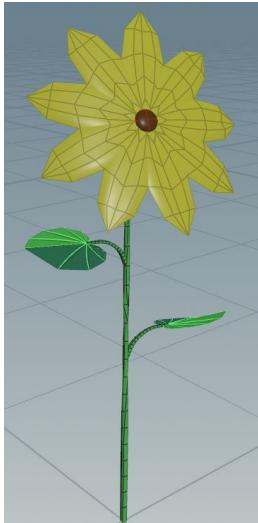
```
Premise !F(0.07)[B]!F(0.07)[E]!F(0.07)A
Rule 1 A=!\\"[B]///[B]///[B]///[B]
Rule 2 B=!(20)\&F(0.05)C
Rule 3 C=!\\"[D]// [D]// [D]// [D]// [D]F(0.04)J
Rule 4 D=!(20)\&F(0.04)J
Rule 5 E=!(20)\^F(0.05)C
```



Listing 2: Flower Two

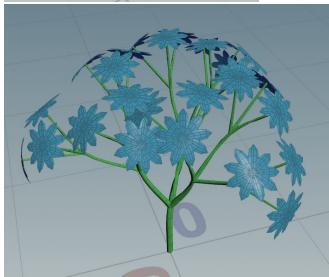
```
Premise F(0.05)A
Rule 1 A = -(15)!F(0.05)[\^BBBBK]/(120)!F(0.03)[\^CCCJM]A
Rule 2 B = -(20)!\&(10)F(0.004)
Rule 3 C = -(10)!\^^(10)F(0.02)
Rule 4 D = -(15)!F(0.05)[\^BBBBK]/(120)[\^CCCJ]A
```

Bei der zweiten Blume bildete sich ein Problem. Sie besteht aus Iterationen, wodurch man durch Verändern der Anzahl der Generationen ein Wachstum sehen kann. Das Problem: die Objekte bilden sich sofort, wodurch es zu einem unschönen Bild wurde. Das neue Blatt und die Blüte hängen ineinander. Durch einfügen einer Linie (Regel A; F(0.03)) sieht es aber akzeptabel aus.



Listing 3: Flower Three

```
Premise !(395)F!/(100)F[CCCK]/(196)F[CCCK]!/(183)F[AAAMJ]
Rule 1 A = -(20)!F(0.02)
Rule 2 B = -(20)!\^(20)F(0.02)
Rule 3 C = -(20)!\&(20)F(0.02)
```



Listing 4: Flower Four

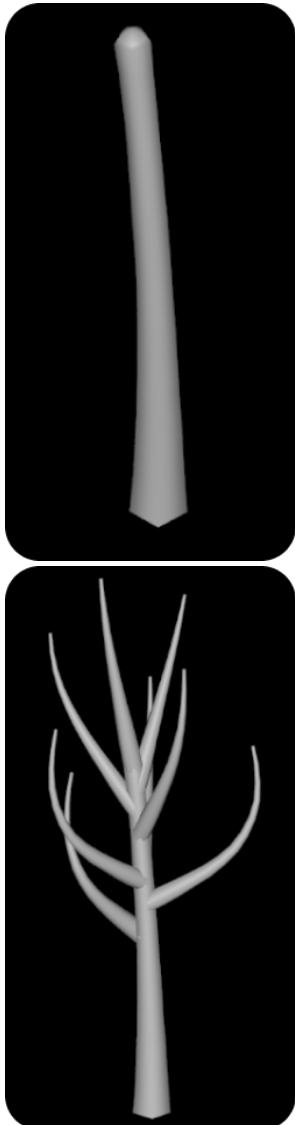
```
Premise FA
Rule 1 A=!"[B]////[B]///BJ
Rule 2 B=!\&FA
```

Mit Hilfe von Gruppen können wir die Blume nun einfärben (color). Erstmal färben wir alles Grün (#21FF26), ohne eine Gruppe anzugeben. Die Blüte und den Kern färben wir durch Angeben der Gruppen lsysX, wobei X der jeweilige Input steht (Blüte = J; Kern = M). An Ende wie immer unser Null Node FLOWER\_OUT.

## Bäume:



## Aufbau:

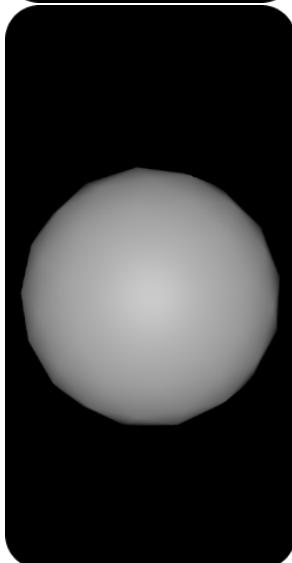


Wir haben uns für den Low Poly Stil entschieden. Deswegen versuchen wir den Baum so simple wie möglich zu erstellen. Am Anfang benötigt man den Tree\_Trunk\_Generator um den Stamm vom Baum zu erstellen. Ich habe mich bei dem Radius für 1.5 und bei der Länge für 20 entschieden. Wenn man den Stamm biegen möchte, kann man dies bei 'Tropism' tun. Den Bend Wert, also wie stark der Stamm gebogen sein soll, habe ich auf 10 gesetzt. Da es nach Low Poly aussehen soll, muss man bei 'Resolution' die Resolution und die Anzahl der divisions runter stellen, damit der Stamm weniger Ecken hat, an denen er gebogen werden kann. Dadurch wird das Ergebnis 'kantiger'.

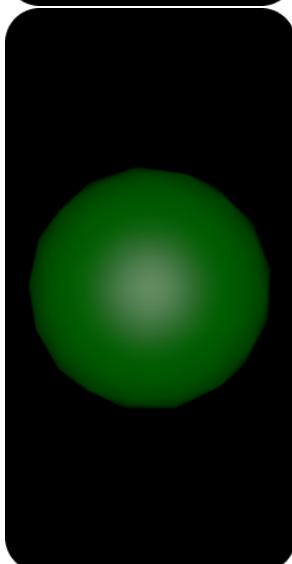
Wir haben jetzt den Baumstamm fertig, und benötigen nun die Äste. Dies kann man mit der Node 'Tree\_Branch\_Generator' erledigen. Die Anzahl der Ästen habe ich auf 8 gesetzt und die Länge auf 0.7. Aber damit die Äste nicht gerade werden, muss man in 'Tropism' bei 'Phototropism' die Gravitation einstellen. In diesem Fall habe ich bei 'Strength' eine positive Zahl (0.11) eingegeben, damit alle Äste nach oben zeigen. Man kann auch eine negative Zahl eingeben, wenn man möchte, dass sie nach unten zeigen.



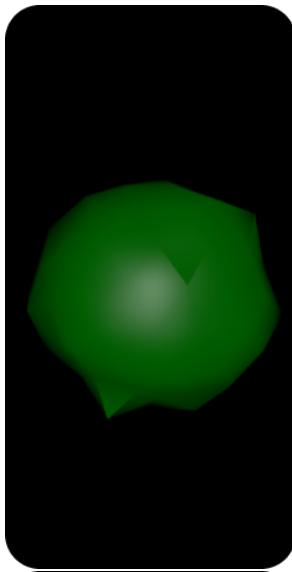
Nun wird der Baum mit der Color Node eingefärbt, damit er nicht weiß bleibt. Ich habe mich für ein Braun mit dem Hex-Code: 351001 entschieden.



Jetzt braucht man eine Sphere Node für die Blätter. Houdini hat auch ein normales Blatt das man benutzen kann. Aber anstatt dies zu benutzen, benutzen wir eine Sphäre, damit der Baum Low Poly artig aussehen wird. In der Node setzen wir den Primitive-type auf 'Polygon'. Dies teilt die Sphäre in mehrere Dreiecke, deren Anzahl man bei Frequency einstellen kann. Ich habe die Frequency zum Beispiel auf 3 gestellt, damit es nicht zu viele und nicht zu wenige werden. Um später, wenn alles zusammen gebaut wird, die Größe einstellen zu können, habe ich eine Transform Node hinzugefügt.

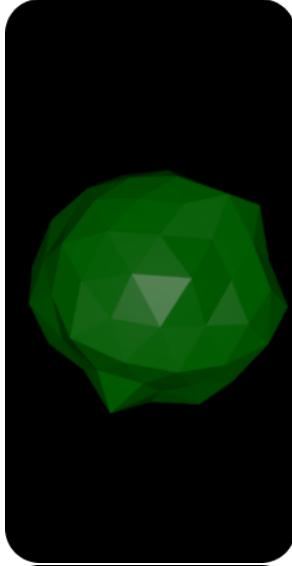


Wie am Anfang wird hier eine Color Node gebraucht, damit die Blätter grün aussehen. Hex Code: 004B00



Danach wird eine Mountain Node über die Sphere gelegt, um es natürlicher aussehen zu lassen. Ich habe folgende Werte gesetzt:

- height: 7.28
- element size: 5.23
- Noise Type: Perlin
- Factal Type: Terrain
- Max Octaves: 9
- roughness: 0.741



Dann wird die Edgecusp Node angewendet damit es mehr nach Low Poly aussieht.



Letztendlich wenden wir die Tree\_Leaf\_Generator Node an. Hier kann man einstellen wie viele Blätter man haben möchte und wo sie plaziert sein sollen. Da wir eine Sphäre für mehrere Blätter haben, sollte man die 'Leaf Node Distance' ziemlich weit auseinander einstellen (bei mir auf 5) damit pro Ast nur eine Sphäre daraufgesetzt wird. Bei 'Size Ramp' stellt man auch ein wo sich die Blätter auf den Ästen befinden sollen. Diese habe ich weit nach rechts gesetzt, damit die Sphäre am Ende der Äste sind.

Der ganze Aufbau des Low Poly Baums sieht mit den Nodes folgendermaßen aus:

