Politecnico di Torino

Corso di Laurea Magistrale in Ingegneria del Cinema e dei Mezzi di Comunicazione

A.a. 2020/2021

Analisi della pipeline di produzione di modelli procedurali in ambiente real-time

Relatore:
Prof. Andrea Bottino

Correlatore:
Dott. Francesco Strada

Candidata:
Alice Amato

A chi porta sulle spalle
il grande peso di un sogno che
va ben oltre le proprie aspettative.

Sommario

L'industria video-ludica è in continua evoluzione e fortemente competitiva. A modellatori e artisti è richiesto di essere tanto competenti, quanto veloci. La ricerca tra ottimizzazione dei tempi e dei costi è una costante del settore e la modellazione procedurale risponde a questa richiesta. Nella modellazione procedurale gli oggetti 3D vengono creati tramite operazioni (traslazioni, estrusioni, rotazioni, copie, etc.), contenenti dei parametri (altezza, lunghezza, angolo di rotazione, numero di copie, ect.). Questi parametri vengono messi in relazione tra di loro attraverso regole, chiamate appunto *procedure*: modificando un parametro si modificano, di conseguenza, quelli ad esso legati. Nella modellazione classica, invece, ogni operazione è fine a se stessa in quanto non sono presenti dipendenze tra parametri.

Il presente lavoro di tesi, pertanto, ha come obiettivo l'analisi di vantaggi e svantaggi nell'utilizzo di una pipeline di produzione, che parte dalla modellazione, passa da una fase di texturing e di creazione di asset e termina con i rendering. L'utilizzo di questa pipeline per la generazione di modelli procedurali in applicazioni real-time risulta vantaggioso, in quanto permette di cambiare l'aspetto dell'ambiente, utilizzando però sempre gli stessi modelli di partenza. Nello specifico, è stata realizzata una stanza arredata e l'utente può apportare modifiche a ciascun elemento della stanza, in base alle procedure scelte in fase di sviluppo. Sono stati utilizzati i software *Houdini*, per la modellazione, e *Unity 3D*, per le rese grafiche ad alta definizione, integrati tramite il plug-in *Houdini Engine for Unity*. Nel corso del lavoro sono state riscontrate alcune criticità, sia al momento dell'integrazione tra i due software, che in Houdini stesso. Queste criticità sono state risolte grazie agli strumenti di scripting offerti da Unity e dal plug-in. Pertanto, la pipeline qui discussa rappresenta un ottimo modello per la realizzazione di oggetti 3D procedurali in ambiente real-time.

Ringraziamenti

In primis, ringrazio il prof Bottino per avermi trovato una tesi, in piena pandemia, quando tutti vacillavano per un futuro troppo incerto, e Francesco Strada, per essere stato infinitamente paziente. Grazie ad entrambi per avermi fatto credere, ancora una volta, in me stessa.

Ad i miei genitori va un grazie speciale. Siete ciò che sognavate di essere e questo è per me l'insegnamento più grande che potevate offrirmi.

A mia sorella, che da sempre rappresenta per me una guida ed una fonte d'ispirazione. È guardando i tuoi passi se oggi sono qui.

A mio fratello, che mi permette di vedere la vita e la felicità in un modo molto diverso rispetto al mio.

A Fabio, perché lo starmi accanto, giorno dopo giorno, è il *miglior* incoraggiamento che tu possa farmi. Grazie per avermi reso una persona migliore.

A Silvia, amica sincera, nonché l'unica con cui io possa essere veramente me stessa. Grazie per aver ascoltato ogni mia lamentela e avermi aiutato a superare le difficoltà con più grinta di prima.

Ringrazio poi tutti gli amici che ho conosciuto in questi lunghi anni a Torino - in particolar modo Francesco, Calogero e Natalia - per aver reso meno pesante la lontananza da casa; e i ragazzi dell'aula studio, per le pause caffè e gli stimoli giusti al momento giusto.

Grazie anche gli amici di giù, quelli di una vita quelli di sempre, per cercare di tenere salda la nostra amicizia, nonostante la lontananza e i cambiamenti inevitabili.

Ringrazio i miei colleghi e rimpiango i mesi che il covid ci ha tolto. Ingegneria del Cinema è stata per me il posto che meritavo.

Grazie soprattutto ad Ernesta e Flavio, presenti in ogni progetto indecente e in ogni momento di sclero e gioia. Non posso pensare, oggi, ad una magistrale senza voi due.

Indice

Elenco delle tabelle

Elenco delle figure

Capitolo 1

Introduzione

1.1 Modellazione procedurale

In computer grafica, per ottenere una geometria voluta bisogna modellare il proprio oggetto, mediante una serie di trasformazioni manuali nello spazio. Per quanto il livello di dettaglio possa essere impeccabile, per modificare l'oggetto creato è necessario intervenire sull'intera geometria, pur volendo perfezionare anche una minima sezione.
La modellazione procedurale, o parametrica, invece, permette di creare oggetti 3D attraverso regole e algoritmi e consente di ottenere più versioni dello stesso modello in tempo reale. Ciò è reso possibile grazie alla definizione di procedure che non interessano necessariamente l'intero processo di modellazione, ma anche solo parte di esso.
Poiché in natura diversi materiali presentano gli stessi pattern che si ripetono, le prime tecniche di modellazione procedurale vennero applicate inizialmente su texture e shader, grazie soprattutto agli studi fatti con il software RenderMan[1]. In un secondo momento, si estese il concetto anche ai modelli 3D, grazie al supporto della potenza delle CPU e GPU di ultima genereazione. [1, 2]
Il modus operandi è diretta conseguenza di quella che in informatica viene chiamata *programmazione procedurale*, dove il codice sorgente viene racchiuso in blocchi che formano delle procedure o subroutine [2]. I programmi scritti con queste regole seguono un modello mentale attraverso step intermedi, fatti di chiamate alle funzioni e rimandi continui. [3]

---

[1]PhotoRealistic RenderMan è il motore di rendering sviluppato dalla Pixar.

[2]Una subroutine, o funzione, è un costrutto sintattico tipico di molti linguaggi di programmazione, come il C, capace di contenere al suo interno sequenze di istruzioni e di restituire determinate valori a seconda dell'input ricevuto.

L'approccio alla modellazione procedurale è, quindi, completamente diverso rispetto alla modellazione manuale. Come prima cosa è bene definire l'oggetto e le sue simmetrie. Poiché queste si ripetono più volte nello spazio, basterà rappresentarne una e poi iterare il modello laddove vi è bisogno. Se, ad esempio, volessimo realizzare una ricostruzione 3D della Torre di Pisa (vedi figura 1.1)[4], le simmetrie che subito risaltano agli occhi sono quelle che vanno dal secondo registro al sesto. Andando sempre più nel dettaglio, poi, si vede come anche le arcate si ripetono. La modellazione dovrà partire proprio da qui.
Modellando una singola parte e non l'intero oggetto si ottimizza sicuramente il tempo impiegato in rapporto al carico di lavoro. Questo è solo uno dei tanti vantaggi che la modellazione procedurale offre. Di seguito ne verranno presentati altri.

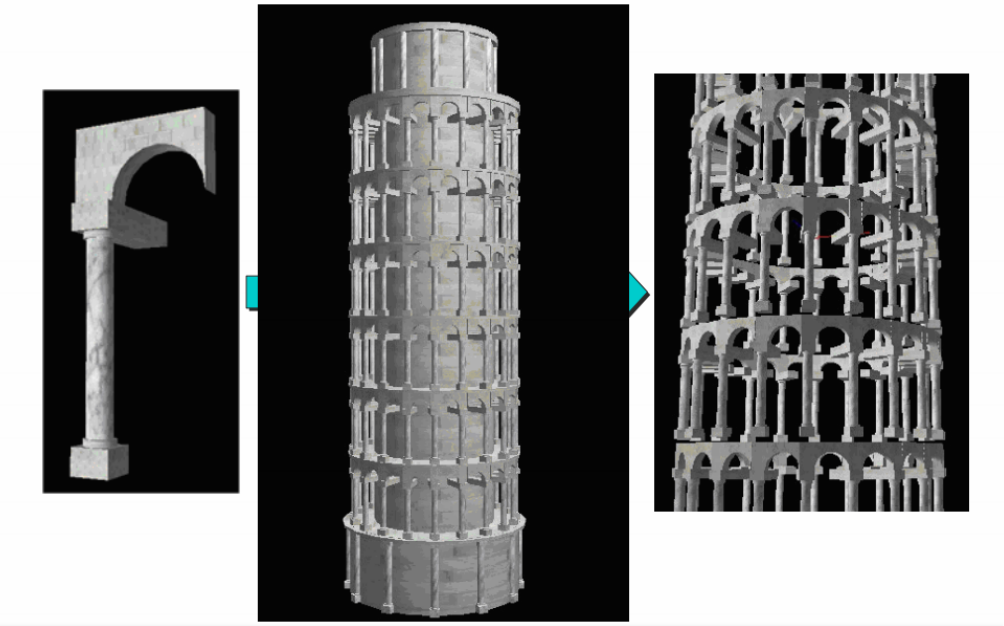


Figura 1.1: Modello procedurale della Torre di Pisa

1.1.1 Vantaggi e svantaggi della modellazione procedurale

Varianti dello stesso oggetto

Consideriamo l'esempio riportato dalla figura 1.1 ed immaginiamo di creare soltanto tre registri, cambiando però spessore, altezza e lunghezza della colonna. A quel punto non avremmo più di fronte a noi la Torre di Pisa, ma un nuovo monumento che condivide con il primo la morfologia.
Variando le procedure che regolano i diversi paramatri dell'oggetto, si possono realizzare altri modelli, fino al non riconoscere neanche quello di partenza. In questo lavoro di tesi è stato creato un modello 3D di un normalissimo bicchiere da tavola, come mostrato in figura 2.15, dove, in seguito alla modifica opportuna di alcuni elementi, si ottiene un calice da vino.
Per raggiungere un oggetto diverso da quello iniziale, il modellatore deve innanzitutto capire che strada intraprendere o cosa collegare attraverso le regole. Questo passaggio può diventare oneroso nel caso di modelli complicati.

Flessibilità

Creare tanti oggetti, e potenzialmente diversi tra di loro, a partire da un solo modello, offre una flessibilità notevole. A proposito dell'esempio appena sopra citato, se ho bisogno di un bicchiere da cocktail nel mio ambiente virtuale, basterà mettere mano su quei pochi parametri interessati, piuttosto che modellare da zero. Nel caso più specifico del software utilizzato per questo progetto, la modellazione è completamente a discrezione del 3D artist. Essendo Houdini - approfondimento al capitolo 2 - di natura nodale, si possono utilizzare diverse gerarchie per ottenere comunque lo stesso modello. Da questo punto di vista, i programmi procedurali sono più adattabili rispetto a quelli classici manuali.

Riduzione di grandi file

Negli ultimi anni stanno trovando largo impiego, anche in situazioni di real-time, i *dataset sintetici*, ovvero l'insieme di tutti quei dati che vengono creati digitalmente tramite algoritmi di *machine learning* e non acquisiti dal mondo reale. Per ricreare ambienti quanto più realistici possibile è necessario un numero elevato di risorse 3D, e questo implica un maggior livello di dettaglio così come una maggiore dimensione della libreria stessa. [5]
Il poter raggiungere più oggetti partendo da uno solo comporta invece un netto ridimensionamento del file che li contiene. Ad esempio, invece di avere una poltrona, un divano a due posti, uno a tre posti e così via, nello stesso archivio, basta avere il modello di uno di questi e aggiungere o sottrarre di volta in volta posti in base alle proprie esigenze.

Ottimizzazione e riduzione budget

La riduzione in termini di byte appena discussa ha come conseguenza diretta l'ottimizzazione anche in fase di rendering. File pesanti richiedono ore e ottime prestazioni delle GPU per essere renderizzati.
Il guadagno in termini di tempo, anzi, è visibile già nella fase antecedente al rendering. Con la modellazione procedurale, il modellatore non è tenuto a ricreare manualmente tutte le parti dell'oggetto o l'interno ambiente, ma solo quegli elementi che vanno legati ad altri tramite regole e procedure. Quindi, in poche parole, il carico di lavoro diminuisce e di conseguenza diminuisce anche il numero di 3d artist impiegati in quello stesso ambito.
Tutto ciò, nel mondo dell'industria dell'intrattenimento, si traduce in termini di riduzione dei costi.

Fattore di casualità intrinseco

Nel mondo reale non è tutto perfetto e lineare. Per dare un senso di veridicità maggiore, spesso bisogna aggiungere del rumore o delle disomogeneità. In particolare, Houdini mette a disposizione una serie di parametri con effetti di casualità e *seed*.

Coinvolgimento maggiore per l'utente

Un obiettivo della tesi era creare un'interazione profonda tra utente e oggetto 3D. In un ambiente realistico digitale dove sono presenti modelli statici, e quindi non parametrici, l'utente si limita ad effettuare operazioni nello spazio, come traslazione o rotazione, in modo quasi passivo. Nel caso di quelli procedurali, invece, il coinvolgimento aumenta in quanto è l'utente a modificare in prima persona i vari oggetti.
Bisogna sottolineare però che non viene data completa libertà, poiché è il modellatore a fissare un range entro il quale è possibile variare i parametri.
Se la modellazione procedurale viene utilizzata per lo sviluppo di videogiochi, *potenzialmente* le esperienze di gioco diventano infinite. Non è detto che tutti i giocatori trovino o utilizzino gli stessi elementi nel gioco, dal momento che questi potrebbero modificarsi. Da questo fattore dipende anche un nuovo modo di vivere la rigiocabilità in quanto la trama potrebbe ristrutturarsi. D'altro canto, però, il programmatore dovrà tenere sotto controllo la generazione procedurale per evitare mondi ripetitivi o non giocabili. [6]

1.2 Stato dell'arte

Con quanto esplicato fin ora, è chiaro che il potenziale della modellazione procedurale è notevole e diventa ancora più interessante in situazioni di *real-time*, come nei videogiochi. M. Bittanti[3] scrive che *"grazie alla generazione procedurale, il contenuto dei giochi cambia laddove la struttura resta uguale"* e il giocatore ha come l'impressione di star vivendo una nuova esperienza nonostante gli ambienti siano *"simili-eppure-leggermente-differenti"*. [8]

Se vogliamo capire quando e come la generazione procedurale ha iniziato a mescolarsi con il mondo dei videogiochi, dobbiamo partire dalle origini, ovvero dai giochi da tavola RPG [4].
Alla fine degli anni Settanta sono stati pubblicati diversi cofanetti di **Dungeons & Dragons** (D&D) [5], tra cui una versione *Advanced* che unificava in tre regolamenti tematici [6] le regole sparse delle versioni precedenti, e un kit aggiuntivo con *dungeon geomorfi* [7]. Quello che era il compito del Dungeon Master nello sviluppo di terreno, successivamente, venne affidato ad un computer, in modo da perfezionare la ri-giocabilità grazie a contenuti adattivi o in modo da diminuire l'authoring. Anche D&D ha quindi vissuto la diretta trasposizione in digitale, con il programma *Dungeon Master's Assistant* che lanciava randomicamente i dadi al posto del DM. [9]

Rimanendo sempre a tema D&D, **Rogue**, classe 1980, è un videogioco fantasy *dungeon crawl*, cioè un gioco di ruolo dove il protagonista, esplorando labirinti (dungeon appunto) casuali fatti di codice ASCII, deve superare diverse difficoltà, pena la morte permanente. Rogue ha dato vita ad una serie di giochi con lo stesso stile, noti come *roguelike*[8]. [10, 11]

---

[3]Matteo Bittanti è un artista, scrittore, curatore, editore e accademico. Sostiene corsi sui *media studies* e *game studies* presso la Libera Università di Lingue e Comunicazione (IULM) di Milano, coordina il Master Universitario di Primo Livello in Game Design e collabora tuttora con l'Università di Stanford.[7]

[4]Dall'inglese role-playing game, il termine sta ad indicare i giochi di ruolo in cui i giocatori vestono i panni di alcuni personaggi e vivono le rispettive avventure.

[5]Si tratta di un RPG fantasy, ideato da Gary Gygax e Dave Arneson. I giocatori si muovono sotto la guida di una sorta di moderatore, chiamato *Dungeon Master* (DM), il quale ha anche il compito di creare il terreno di gioco.

[6]Ovvero Monster Manual, Player's Handbook e Dungeon Master's Guide.

[7]Questo kit conteneva 10 piastrelle modulari di dimensione diversa, che potevano essere collegate tra di loro per creare svariati dungeon.

[8]Il termine fu coniato nel 2008 durante l'International Roguelike Development Conference di Berlino, dove sono state messe a punto le regole base per tutti quei giochi che volessero proclamarsi

Figura 1.2: Mike gioca a D&D in Stranger Things

Tra i primi videogiochi - roguelike a parte - ad usare elementi procedurali troviamo **The Sentinel** ed **Elite**, disponibili entrambi anche per Commodore 64[9].
Nel primo, il protagonista, Synthoid, è un androide che deve sfuggire ad una sentinella poiché questa assorbe la sua energia. L'obiettivo è quello di prendere la posizione della sentinella in modo da passare al livello successivo. I livelli totali sono 10.000 e il terreno di gioco è generato casualmente.
Elite è un gioco di genere gestionale[10] e simulazione spaziale in un universo formato da 8 galassie, ciascuna con 256 sistemi solari diversi con un numero di pianeti in orbita che va da 1 a 12. Alcuni di questi sistemi però non permettevano una vera e propria esplorazione in quanto l'astronave non poteva atterrarvi. Inoltre, il generatore che doveva affibbiare nomi del tutto casuali a pianeti, a volte, pescava frasi non del tutto consone. [12]

La *procedural content generation* (PCG) trova quindi la sua massima espressione nella ricostruzione di tutti quegli ambienti che devono essere esplorati per permettere al gioco di andare avanti, come accade per gli *open world* e *open universe*.
Un esempio più recente in quest'ultimo campo ci è offerto da **No Man's Sky**, un'avventura dinamica[11] di Hello Games. I giocatori vivono le esperienze di colonizzatori spaziali e vanno di pianeta in pianeta al fine di poterlo occupare. Il

"figli di Rogue".

[9]Il C64 è un microcomputer degli anni '80, con una RAM da 64kB e microprocessore MOS 6510.

[10]O manageriale, ne fanno parte tutti quei videogiochi che imitano le azioni della vita quotidiana.

[11]Genere che mischia elementi tipici dell'avventura con l'azione.

videogioco presenta $18 * 10^{30}$ pianeti diversi tra loro, creati grazie ai principi della *superformula*[12], poi resa procedurale. [13]

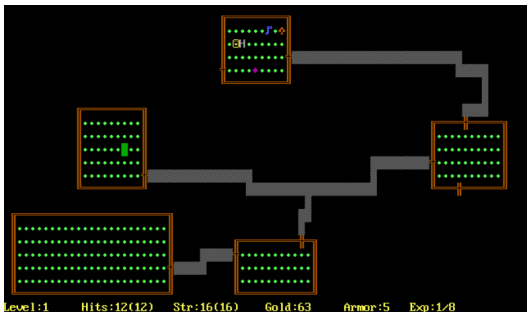


Figura 1.3: Interfaccia del gioco Rogue

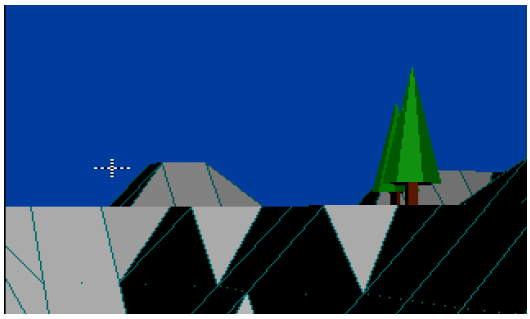


Figura 1.4: Interfaccia del gioco The Sentinel

Figura 1.5: Interfaccia del gioco Elite

Figura 1.6: Interfaccia del gioco No Man's Sky

La generazione procedurale facilita senza ombra di dubbio il *game artist* nell'elaborazione di tutti i livelli di gioco, ma ciò non toglie il fatto che la si può trovare comunemente per gli oggetti più caratterizzanti del gioco, come accade per il milione di armi di **Bordelands**. Si tratta di un videogioco sparatutto in prima persona ed action RPG, sviluppato da Gearbox Software e pubblicato nel 2009 da 2K Games. Il concept designer del sequel del videogioco, Kevin Duc, ha spiegato che, avendo otto produttori diversi, ognuno con un proprio gusto, ha *"dovuto dargli uno stile visivo e descriverlo attraverso le forme, i colori, fare in modo che le armi assomigliassero al modo in cui avrebbero giocato"* per *"far sentire tutti i diversi giocatori fedeli ad una particolare marca"*. [14]

[12]Permette di ricavare le cosiddette superforme, a partire da funzioni circolari in due dimensioni.

Figura 1.7: Interfaccia del gioco Borderlands

Spesso, tutto ciò che riguarda il mondo videoludico ha a che fare con il mondo del cinema. Troviamo anche esempi di modellazione procedurale negli eserciti de *Il signore degli anelli*[13].

[13]Trilogia diretta da Peter Jackson ed ispirata ad uno dei romanzi fantasy più famosi, scritto da J. R. R. Tolkien.

1.3 Pipeline di produzione

In questo lavoro di tesi sono state messe in campo tecniche di modellazione procedurale e testate in ambito real-time. Per la realizzazione è stata seguita una pipeline ben precisa, che possiamo racchiudere in 5 fasi.

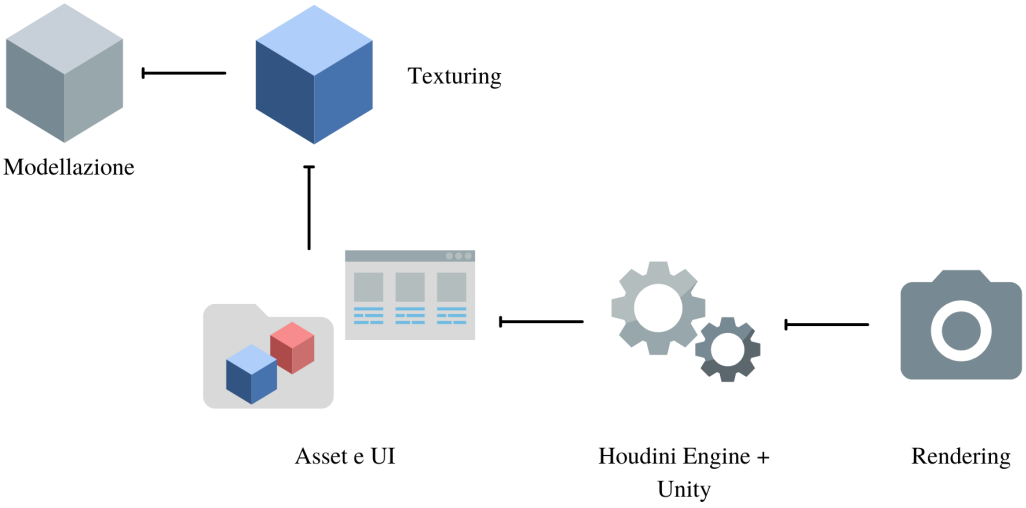


Figura 1.8: Schema della pipeline di produzione

1. **Modellazione**
   Per la creazione dei modelli 3D è stato utilizzato il software Houdini che, grazie alla sua natura nodale, consente di modellare in modo procedurale.

2. **Texturing**
   Sempre su Houdini sono stati applicati gli *shaders* e l' *UV-map*. In particolare, sono stati utilizzati i materiali gratuiti di Quixel[14]. Tra i suoi prodotti, Bridge è quello che permette un collegamento diretto con Houdini e Unity: basta semplicemente inserire il package all'interno dei due software e settare poi le impostazioni corrette per l'export del materiale scelto.

3. **Asset e UI**
   Ciò che è stato realizzato in Houdini è stato ulteriormente impacchettato all'interno di asset. Oltre a contenere la geometria ed i materiali, questi asset

[14]Ormai di proprietà di Epic Games, Quixel è un'azienda che si occupa di librerie di risorse 3D, gratuite e non.[15]

servono anche a creare un'interfaccia utente con le specifiche dei parametri procedurali.

4. **Real-time**
   Grazie al plug-in Houdini Engine for Unity è stato possibile testare il comportamento procedurale in un ambiente 3D real-time, controllato dal software Unity. Qui sono emerse alcune criticità, trattate più a fondo nel corso di questa tesi.

5. **Rendering**
   Variando i parametri dell'asset, l'oggetto in questione cambiava forma; ciò significa che la stanza, all'interno della quale sono stati posti i diversi oggetti, cambiava d'aspetto. Per mettere a confronto questi diversi preset, infine, sono stati effettuati dei rendering su Unity.

Capitolo 2

Proceduralità in Houdini

Sviluppato dalla canadese Side Effects Software, Houdini rappresenta il software numero uno nell'ambito della modellazione procedurale. La sua caratteristica principale risiede nel *paradigma procedurale*, ovvero nella creazione di una rete di nodi collegati tra di loro sottoforma di struttura ad albero.
In informatica, un albero è composto da un nodo radice, chiamato padre, e da uno o più sotto-nodi, chiamati figli, i quali ereditano proprietà ed informazioni dal padre. Nella figura 2.1 il nodo più scuro rappresenta il padre.

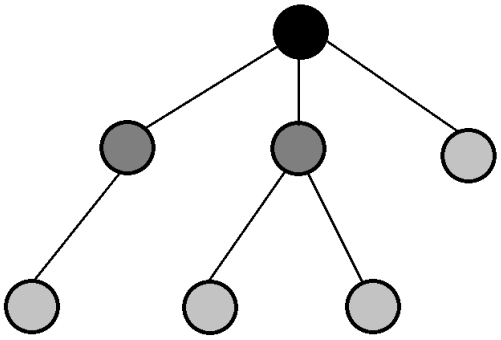


Figura 2.1: Esempio di una struttura ad albero

Il modello 3D che vediamo come risultato finale altro non è che l'unione di più reti. Ciascuna rete assolve a compiti diversi, così come ciascun nodo si riferisce ad una funzione specifica. E' possibile muoversi all'interno delle reti, come mostrato in figura 2.3, per vederne il contenuto, aumentare lo zoom, selezionare un nodo piuttosto che un altro e metterli in collegamento.

Versione	18.5
Build	408
Licenza	Educational

Tabella 2.1: Versione di Houdini utilizzata

Come funziona Houdini

Per comprendere il funzionamento di Houdini, partiremo analizzando gli elementi che compongono l'interfaccia utente (UI) della figura 2.2.
L'oggetto 3D che si vuole trattare è visibile nella finestra della **vista** e gli **strumenti** utili alla modellazione sono presenti nello **scaffale**, mentre quelli più comuni - traslazione, rotazione, etc. - in prossimità di questa.
A parte i classici **controlli** per la **visualizzazione** o la **playbar** per le animazioni, ciò che salta subito all'occhio riguarda l'**editor dei paramatri** e l'**editor di rete**. Il primo permette di modificare, appunto, i parametri del nodo che è stato selezionato nella rete del secondo editor. Possiamo immaginare le reti come le cartelle del nostro computer e i nodi come i file.[16]

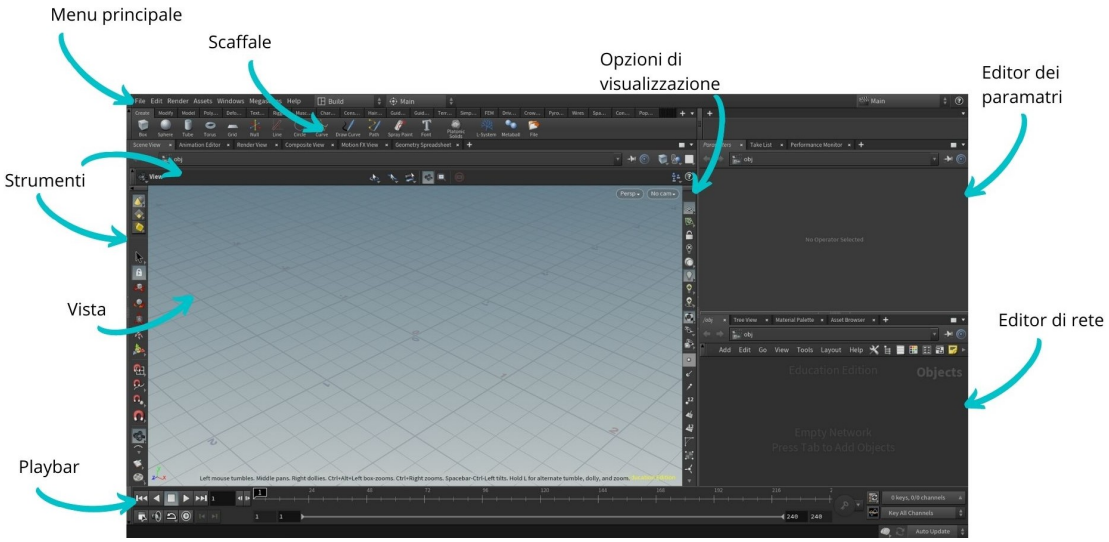


Figura 2.2: Interfaccia utente di Houdini

Gli elementi che compongono la scena, dalle luci ai modelli geometrici, sono contenuti all'interno della rete oggetto (**OBJ**), che rappresenta quindi il guscio più esterno. I collegamenti permettono di trasmettere le proprietà in output del padre all'input dei figli, secondo l'organizzazione della gerarchia; se ruoto il genitore, si muoverà di conseguenza anche il figlio. Per accedere ai singoli nodi che danno la

geometria al modello (figura 2.3), basta cliccare sull'oggetto Geometry (**SOP**, cioè Surface Operator) nella rete OBJ.

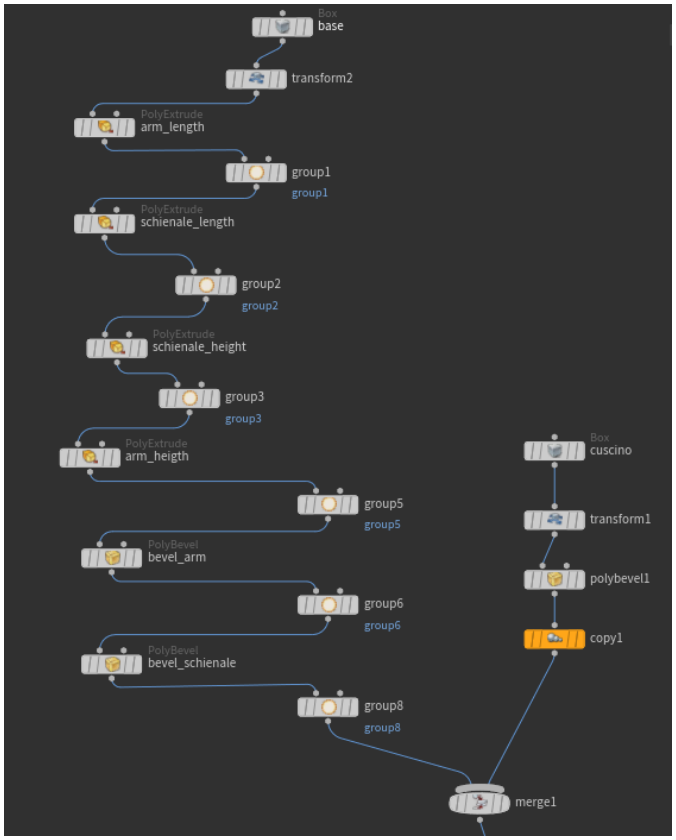


Figura 2.3: Editor di rete

Un'altra rete utilizzata per il progetto di tesi è la cosiddetta rete **VOP, cioè VEX Operator**, in grado di contenere al suo interno i nodi riguardanti texture e materiali.

2.1 Modellazione

Prima di parlare di modellazione vera e propria, è bene fare una panoramica su quelle che in Houdini vengono chiamate *primitive*. Si tratta di entità geometriche che si collocano a metà tra il punto e un oggetto e le principali sono:

- **Spline di NURBS e di Bezier**: curve che possono essere modellate attraverso vertici di controllo (CV).

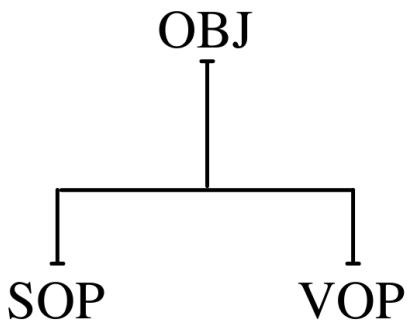


Figura 2.4: Schema delle reti

- **Poligoni e mesh**: i primi sono figure geometriche costituiti da vertici e definiti entro dei bordi fissi; i secondi, invece, possono poi essere convertiti in NURBS e questo permette di lavorarli con più facilità.

- **Polygon soup**: come suggerisce il nome stesso, è un miscuglio di poligoni in grado di occupare un minor spazio in memoria, utile soprattutto in fase di rendering o di cache. Pertanto, maggiore è il numero di poligoni presenti nella *zuppa*, maggiore è il vantaggio. Per ottenere una polygon soup, basta utilizzare il nodo *Polysoup*.

- **Packed Primitive**: producono la geometria voluta al momento del rendering grazie alle informazioni contenute al loro interno. Sono molto utili per modelli complessi e pesanti.

- **Metaball**: due superfici sferiche centrate attorno ad un insieme di punti e, se messe l'una accanto all'altra, le due sfere si uniscono (vedi figura 2.5).

- **Forme quadratiche**: come sfere/ellissoidi, cerchi/ellissi e tubi/coni, caratterizzati da parametri matematici, quali altezza e lunghezza, e da un vertice posto al centro, come si evince dalla figura 2.6.

Adesso possiamo entrare nel cuore della modellazione in Houdini. Per poter lavorare su un oggetto, come prima cosa bisogna crearlo. Dal tab in alto, nella sezione *Create*, è possibile selezione il tipo di geometria, a scelta a partire dai grandi classici come cubo, cilindro o sfera, sino a L-system o metaball. In questo modo viene impostato il livello scena e si può passare alla rete SOP dove modellare veramente.

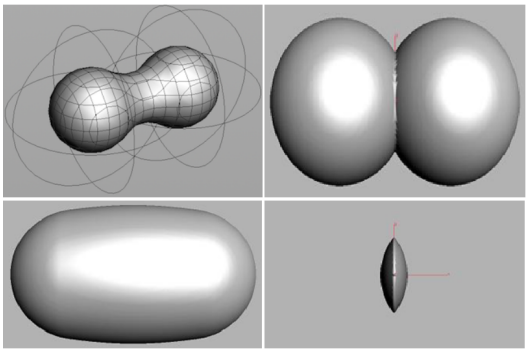


Figura 2.5: Metaball

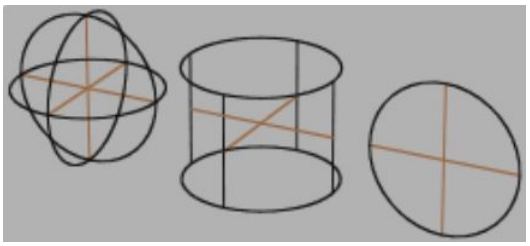


Figura 2.6: Forme quadratiche

2.1.1 Come rendere procedurale la modellazione

A ciascun nodo vengono assegnati dei parametri, dipendenti dalla natura stessa del nodo. Ad esempio, un cubo ha proprietà inerenti alla posizione, alla scala e alla rotazione, mentre un cilindro, in aggiunta, presenta altezza, numero di righe e colonne che lo compongono. Queste proprietà possono essere modificate nell'apposita finestra di editor.
Grazie alla struttura procedurale di Houdini, i parametri dei vari nodi possono dipendere da altri parametri di altri nodi attraverso l'uso di *espressioni*. Le espressioni sono brevi comandi, tipicamente scritti in Python o VEX[1], che calcolano i valori matematici assegnati ad un parametro, modificandolo di conseguenza.

Per capire com'è possibile rendere procedurale la modellazione, di seguito saranno presentati i vari modelli 3D realizzati nel lavoro di tesi ed in che modo sono stati creati.

Divano

Il sofa che vedete nella figura 2.7 non è statico, cioè non ha sempre questa conformazione. Innanzitutto si possono modificare le altezze e le dimensioni dello schienale e dei braccioli. Per poterlo fare, è necessario controllare i fattori di scala nelle direzioni di orientamento opportune. Inoltre, per non avere il modello troppo squadrato, i rispettivi vertici agli angoli possono essere smussati a

[1]VEX è un linguaggio di alto livello, basto sulla sintassi del C e del C++. In Houdini è molto utilizzato soprattutto nella scrittura degli shader.

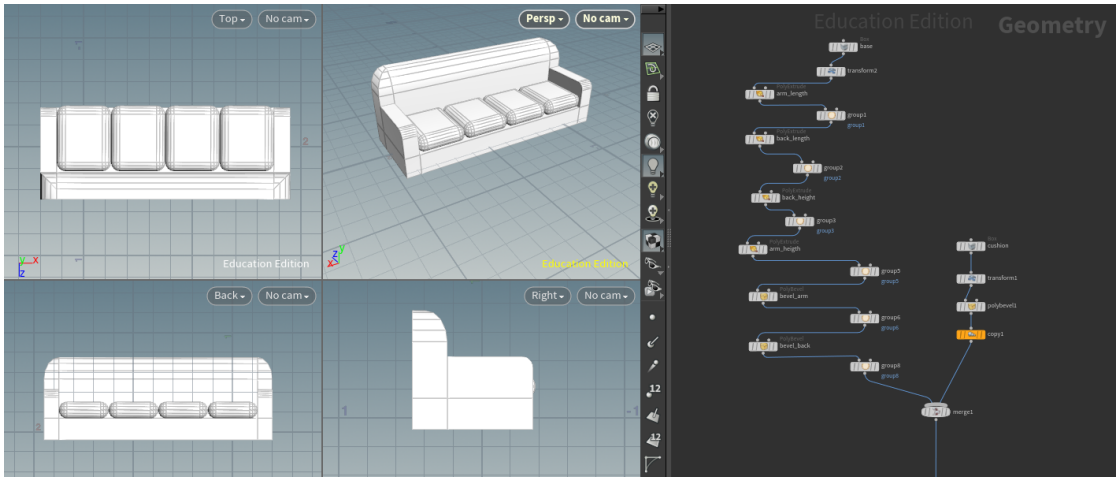


Figura 2.7: Modello 3D del sofa

proprio piacimento. Houdini mette a disposizione un nodo, il *Poly bevel*, che, come suggerisce il nome stesso, permette di arrotondare il modello effettuando l'operazione sul gruppo di vertici prima selezionati. In particolare, si può scegliere il tipo di forma da dare allo smussamento (in questo caso è stato selezionato *Round*).

Al fine di essere quanto più funzionale all'ambiente circostante, si può decidere il numero di posti del sofa, passando da una poltrona ad un divano a due o più posti. Questa scelta viene applicata poi sul nodo *Copy and transform* e andrà ad influenzare il numero di copie dei cuscini; questi, oltre ad essere incrementati, subiranno una traslazione lungo l'asse x, dettata dalla loro dimensione in x.
Ciò da solo, però, non basta: anche il resto del divano deve comportarsi di conseguenza. Se proviamo ad analizzare il nodo che riguarda le trasformazioni del cuscino con quello del resto del divano (vedi codici sotto), notiamo che esiste una forte dipendenza tra i due. I fattori di scala in z del cuscino dipendono da quelli del divano, mentre le dimensioni in x del secondo sono collegate a quello del primo insieme al numero di copie.

Dimensione in Z del cuscino (transform1):

```
1    ch("../transform2/sz") + 0.004
```

Traslazione in X del divano (transform2):

```
ch("../transform1/sx") / 2 * (ch("../copy1/ncy") - 1)
```

Dimensioni in X e Z del divano (transform2):

```
(ch("../transform1/sx") * (ch("../copy1/ncy")) + 0.06
ch("../sz") - 0.4
```

Parametri procedurali
Numero di posti
Larghezza dei braccioli
Altezza dei braccioli
Arrotondamento dei braccioli
Larghezza dello schienale
Altezza dello schienale
Arrotondamento dello schienale
Lunghezza del cuscino
Altezza del cuscino
Arrotondamento del cuscino

Tabella 2.2: Parametri procedurali del divano

Libreria

Quello appena presentato è uno dei tanti modi di fare copia e incolla su Houdini. Un procedimento abbastanza semplice prevede l'utilizzo di un ciclo *for each*. Poiché le espressioni usate per rendere procedurali i nodi possono essere scritte tramite linguaggi di programmazione, non c'è da stupirsi se esistono anche cicli o, più in generale, strutture tipiche dell'informatica di base. L'immagine 2.8 riporta come sono stati impilati gli scaffali e come viene impiegato il ciclo for-each, evidenziato in arancione. All'interno del blocco si trova il nodo *Copy to Points* che crea delle

copie della geometria ricevuta in ingresso, ovvero un singolo scaffale. A fare da padre al ciclo è, invece, una semplice linea verticale, i cui punti determinano il numero dei ripiani e la cui lunghezza dipende da questi punti e dalla dimensione in y dello scaffale, secondo la regola:

```
1    ch("points") - ch("../transform1/sy")
```

Lo scaffale invece è stato ottenuto tramite trasformazioni ed estrusioni a partire da un cubo.

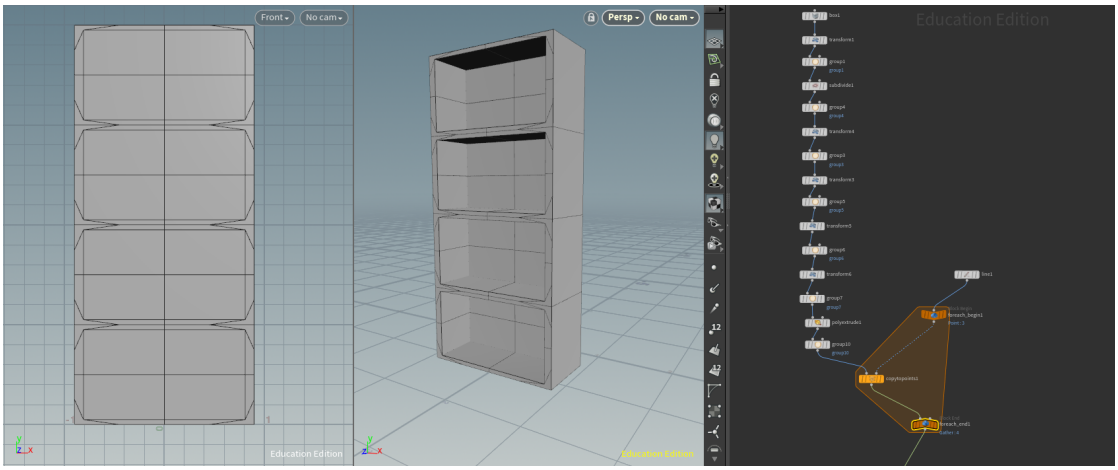


Figura 2.8: Modello 3D della libreria

Parametri procedurali
Numero degli scaffali
Larghezza
Lunghezza

Tabella 2.3: Parametri procedurali della libreria

Lampada

La lampada, mostrata in figura 2.9, è stata realizzata grazie all'unione di tre parti diverse: la base e il cilindro portante, il paralume e le aste su cui poggia e infine la lampadina. Analizzeremo questi elementi singolarmente per capirne meglio il funzionamento (2.10).

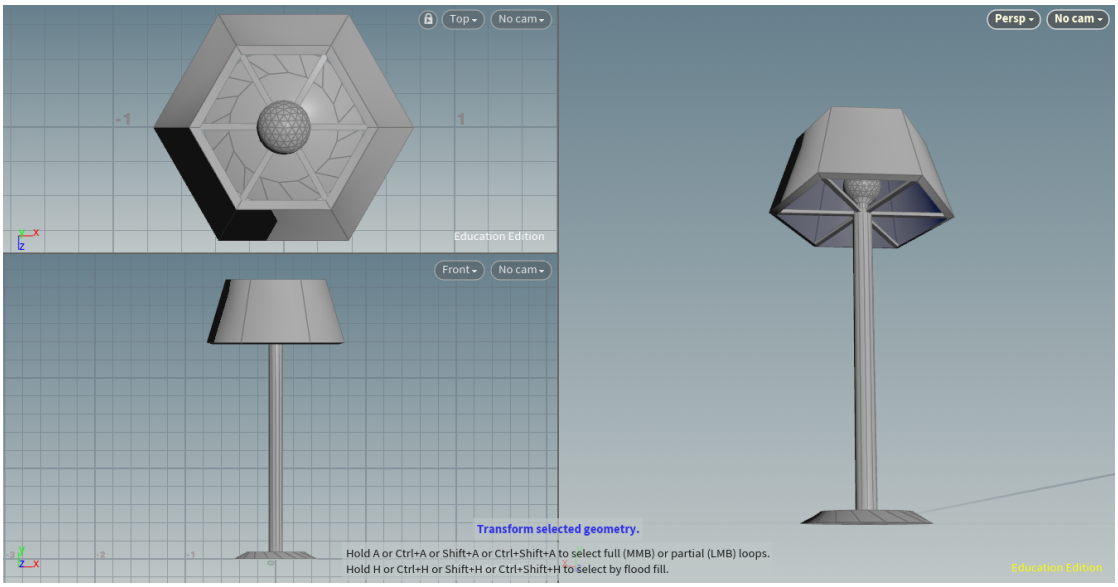


Figura 2.9: Modello 3D della lampada

1. **Base e cilindro portante**
   Per la realizzazione della base si è partiti da un *Circle* di tipo poligono che poi è stato opportunamente estruso per ottenere una forma solida in stile cono. Le dimensioni e i lati della base sono parametri procedurali e pertanto possono essere modificati dall'utente.
   Per quanto riguarda il tubo, questo altro non è che un cilindro, dalla cui altezza dipende la traslazione in y di ciò che sta sotto e di ciò che sta sopra. Le espressioni utilizzate sono rispettivamente:

```
1    (ch("../asta1/height") / 2) - ch("../asta1/height")
2
3    ch("../asta1/height") / 2
```

2. **Paralume e aste**
   La peculiarità del paralume è data dalla possibilità di scegliere, oltre alle sue

dimensioni e inclinazione, anche la sua struttura: da quadrata a circolare. Da quest'ultima è ovvio che dovrà dipendere il numero di asticelle che lo sorreggono, poiché dal centro del cilindro portante si diramo in direzione di ciascun vertice del paralume stesso. Per ovviare a questo problema, si è scelto come forma massima quella di un decaedro e come minima quella di un cubo. Ma come dire al modello 3D di modificare di conseguenza il numero delle aste? Houdini mette a disposizione un nodo, lo *Switch*, che accetta in ingresso n input e, tramite dei settaggi, ne manda uno in output. In questo caso, il controllo da effettuare può essere descritto come

```
1        ch("../paralume/divs") - 4
```

dove il -4 serve a riportare la numerazione esatta, in quanto lo *Switch* parte da 0.

3. **Lampadina**
   Infine, la lampadina è stata modellata utilizzando una sfera, per la parte del globo di vetro, e un cilindro, per la parte di avvitamento. Per diventare un unico modello 3D, le due primitive sono state poi unite con il nodo *Merge*.

Parametri procedurali
Altezza
Larghezza della base
Forma della base
Altezza della base
Forma della base
Forma del paralume
Altezza del paralume
Arrotondamento del paralume

Tabella 2.4: Parametri procedurali della lampada

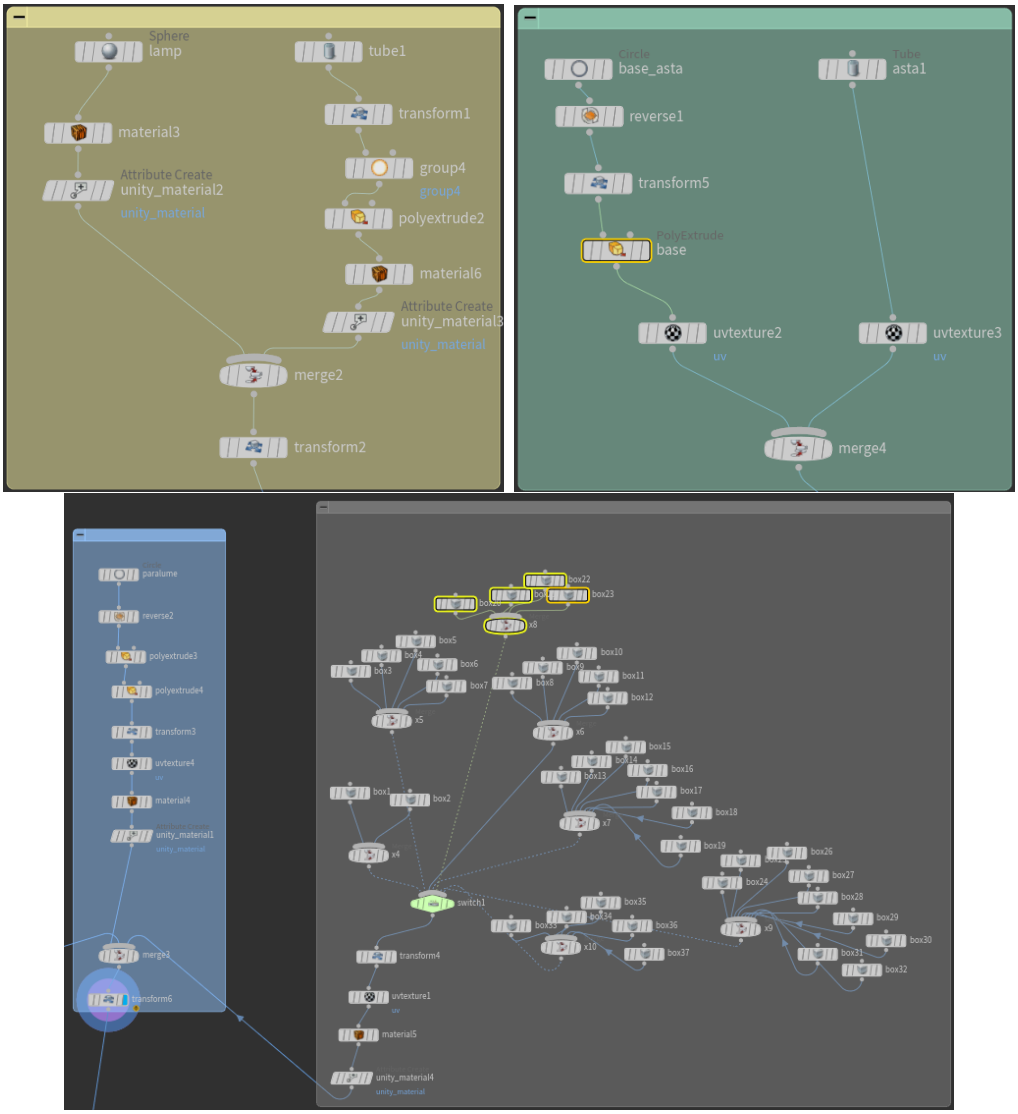


Figura 2.10: Parti costituenti il modello della lampada

Tavolo

Se pensiamo come rendere procedurale un tavolo, la prima cosa che ci viene in mente che l'utente possa variare è l'altezza. Le gambe sono state create partendo da un linea, avvolta dalla mesh modellata dal nodo *Sweep*. Tutto ciò è stato poi posto agli angoli di una griglia, formata da due righe e due colonne, usando il *Copy to points*. La griglia estrusa rappresenta il piano del tavolo, che può essere opportunamente scalato in X e in Z. In questo modo, aumentando la lunghezza della linea, aumenta la lunghezza delle gambe e il piano del tavolo si muove di

conseguenza.

Gli altri parametri modificabili sono:

Parametri procedurali
Lunghezza
Larghezza
Altezza
Altezza dell'estrusione del piano
Forma del piano
Forma delle gambe
Larghezza delle gambe

Tabella 2.5: Parametri procedurali del tavolo

Attributi come la forma permettono di avere un tavolo rotondo o perfettamente squadrato o con i bordi leggermente inclinati.

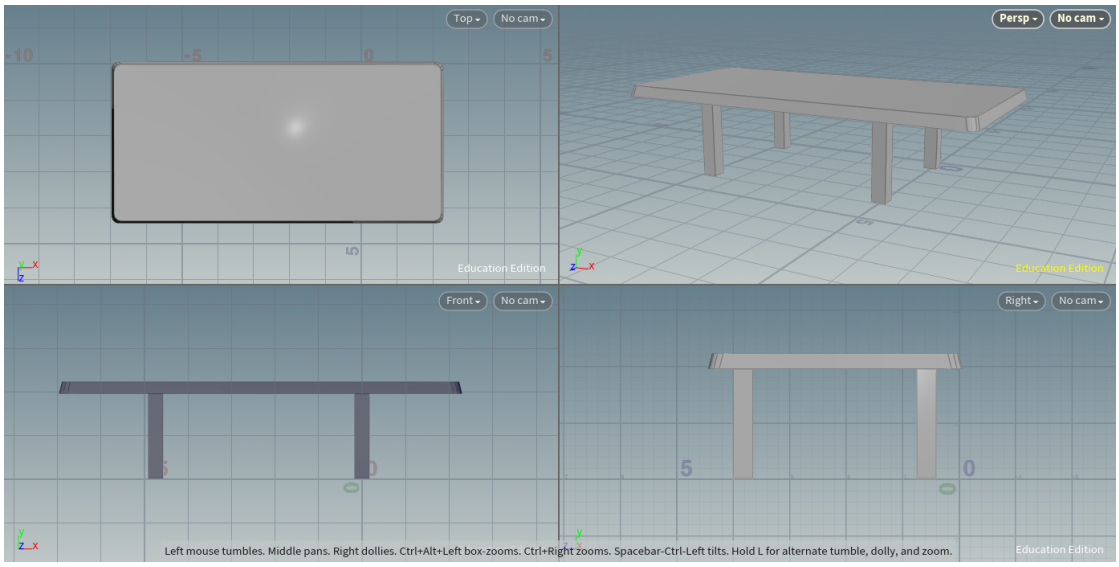


Figura 2.11: Modello 3D del tavolo

Bottiglia

Di bottiglie diverse ne esistono veramente tante, per cui l'obiettivo era quello di rendere più parti parti procedurali. La figura 2.12 mostra quattro diversi modelli che si possono raggiungere.

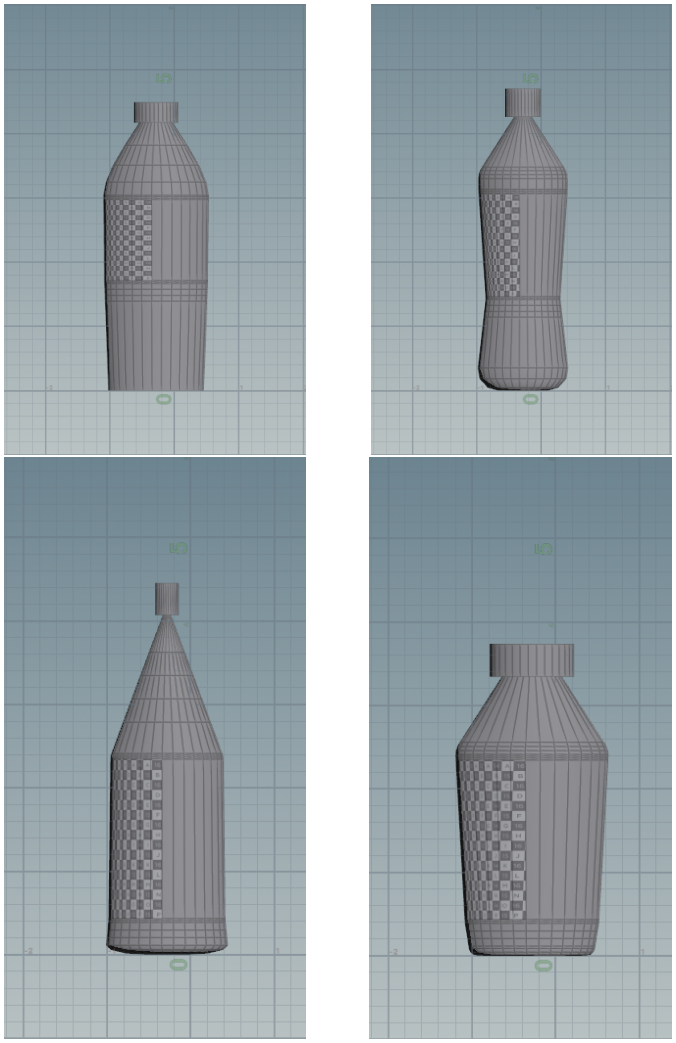


Figura 2.12: Bottiglie ottenibili dallo stesso modello

Partendo dall'alto, il tappo può essere ridimensionato in larghezza e altezza. Stessa cosa per il resto della bottiglia, oltre al fatto che si possono impostare altri parametri, come l'arrotondamento o la forma. Infine, si può scegliere tra un fondo più spigoloso o uno dai lineamenti più morbidi.

La modellazione è avvenuta usando un'unica primitiva e applicando le trasformazioni del caso. Quest'approccio, quindi, non ha richiesto l'utilizzo di particolari espressioni.

Poco sopra, è stato analizzato l'oggetto tavolo, destinato ad ospitare il modello della bottiglia. Per evitare la noia all'utente nell'inserimento di più bottiglie, tra i parametri procedurali, disponibili nella tabella 2.6, sono stati inseriti la dimensione del tavolo e un coefficiente di casualità. Basterà, allora, immettere i fattori di scala del tavolo su cui verranno poste le bottiglie, mentre il secondo termine assegna la posizione e il numero, del tutto random, delle bottiglie. L'immagine 2.13 porta un esempio di come Houdini gestisce questa distribuzione.

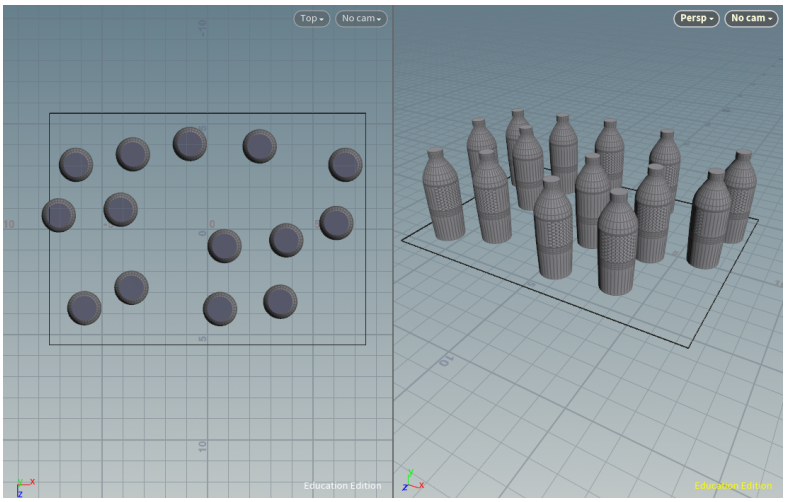


Figura 2.13: Disposizione casuale delle bottiglie

Per ottenere quanto appena descritto, innanzitutto bisogna creare più copie dello stesso modello. E' stato impiegato un costrutto di tipo *for-each* contente un *Copy stamp*. A fare da padre al ciclo è un cubo, le cui dimensioni in X e Z sono le stesse di quelle del tavolo. Non è stata contemplata l'altezza in quanto il piano deve possedere al suo interno dei punti corrispondenti alla posizione delle bottiglie, poste tutte alla stessa altitudine. Il nodo *Points from volume* genera una serie di punti all'interno del volume che riceve in ingresso. Per non avere le bottiglie sovrapposte, è stato un inserita una separazione tra i punti pari a:

```
1    ch("../transform2/sx") + 2.2
```

dove la dimensione in X è quella delle bottiglie stesse.
Inoltre, la casualità è dettata dal parametro *Jitter scale*, inserito tra i parametri procedurali.

Parametri procedurali
Altezza
Larghezza
Altezza del logo
Parte superiore - altezza
Parte superiore - larghezza
Parte superiore - forma
Parte superiore - arrotondamento
Tappo - altezza
Tappo - larghezza
Fondo
Dimensioni tavolo
Casualità

Tabella 2.6: Parametri procedurali della bottiglia

Quanto fatto finora non è abbastanza. Abbiamo una bottiglia che viene copiata ed incollata in diversi punti del tavolo, ma ognuna di queste copie è identica all'originale, anche in traslazione e rotazione. Nella vita reale tutto ciò desterebbe un minimo sospetto. Come ultimo nodo nella gerarchia di modellazione troviamo un *Transform*[2], il cui compito è, appunto, quello di gestire un'ulteriore possibilità randomica. Le regole utilizzate per la traslazione in X e Z e per la rotazione in Y sono rispettivamente:

```
1    fit01(rand(detail("../foreach_begin1_metadata1/", iteration, 0)
     +5000), -1, 1)
2    fit01(rand(detail("../foreach_begin1_metadata1/", iteration, 0)
     +5000), -1, 1)
```

```
1    fit01(rand(detail("../foreach_begin1_metadata1/", iteration, 0)
     +1564), -180, 180)
```

[2]Si tratta del *Transform2* utilizzato nell'espressione 2.1.1.

Le espressioni utilizzate in tutte e tre le righe restituiscono:

- *detail*: il valore di un *detail attribute*, in questo caso l'iterazione estrapolata dai metadati del ciclo for-each
- *rand*: un numero pseudocasuale compreso tra 0 e 1
- *fit01*: un numero compreso tra gli ultimi due parametri, che rappresentano quindi un minimo e un massimo (-1/1 nel caso della traslazione, e -180/180 per la rotazione)

Tazza

Come messo in evidenza dalla figura 2.14, si è partiti nella modellazione da un poligono di tipo cerchio, orientato sul piano ZX, a cui sono seguite quattro operazioni di estrusione. La prima per definire l'altezza, le altre due per ottenere il fondo, e infine, l'ultima, per la base di appoggio della tazzina. In questo modo, però, aumentando l'altezza della parete esterna, quella interna rimarrebbe fissa, quando in realtà dovrebbe andare in direzione opposta. Le due parti sono state collegate nel seguente modo:

```
1    - ch("../heigth/dist") + 0.1
```

Il manico non è stato modellato a parte e unito al resto della mesh con un *Merge*, ma agendo sul modello della tazza stesso. Il nodo *Knife* può essere impiegato per disegnare delle linee che, appunto, taglino la geometria. Sono state effettuate due incisioni in posizione dell'aggancio del manico, smussate poi grazie al nodo *Poly bevel*. Le linee sono quindi passate da due a quattro. Houdini mette a disposizione un nodo, *Poly bridge*, che costruisce una superficie poligonale tra i bordi sorgente e di destinazione, ed in grado di fondersi al resto dell'oggetto.

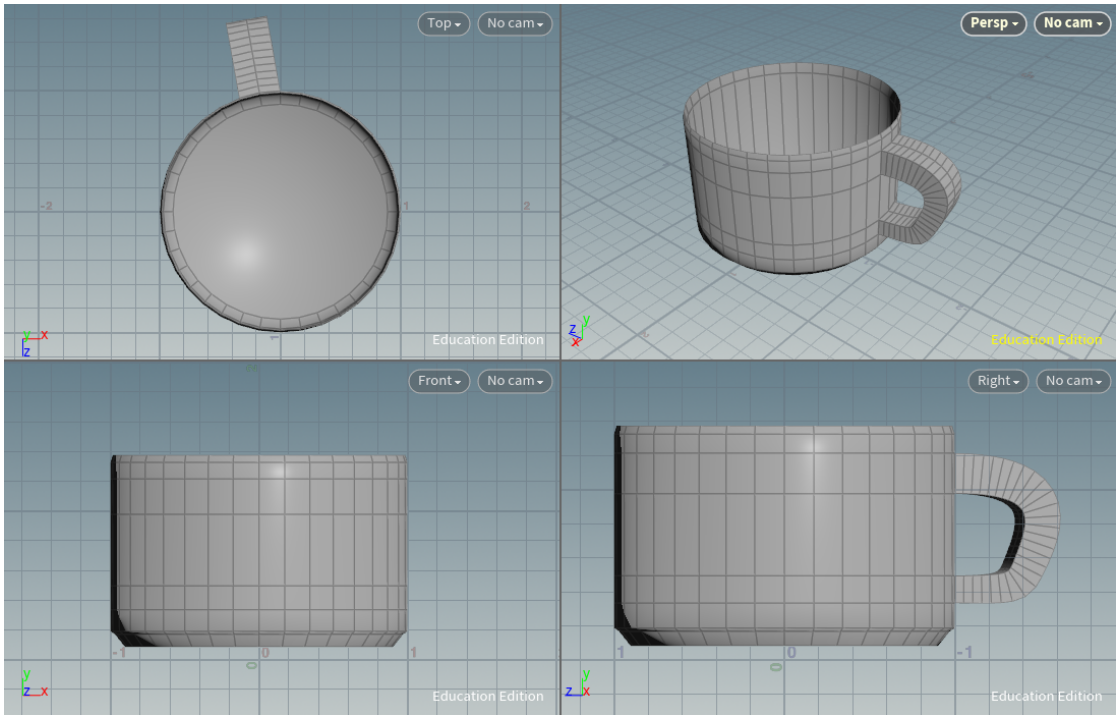


Figura 2.14: Modello 3D della tazza

Bicchiere

Il bicchiere è stato modellato un po' come la tazza, ovvero iniziando con un cerchio e poi facendo trasformazioni ed estrusioni. Il fatto che Houdini permetta di lavorare su singole parti del modello ha permesso di ottenere un oggetto altamente procedurale.

Nella tabella 2.9 sono elencati i paramatri che l'utente può modificare, mentre l'immagine 2.15 mostra i diversi bicchieri che si possono avere.

Parametri procedurali
Altezza
Forma
Smussamento
Altezza della base
Larghezza della base
Forma del manico
Larghezza del manico

Tabella 2.7: Parametri procedurali della tazza

Parametri procedurali
Larghezza
Altezza
Forma
Forma della coppa
Altezza della coppa
Arrotondamento
Larghezza della gamba
Altezza della gamba
Altezza della base
Forma della base

Tabella 2.8: Parametri procedurali del bicchiere

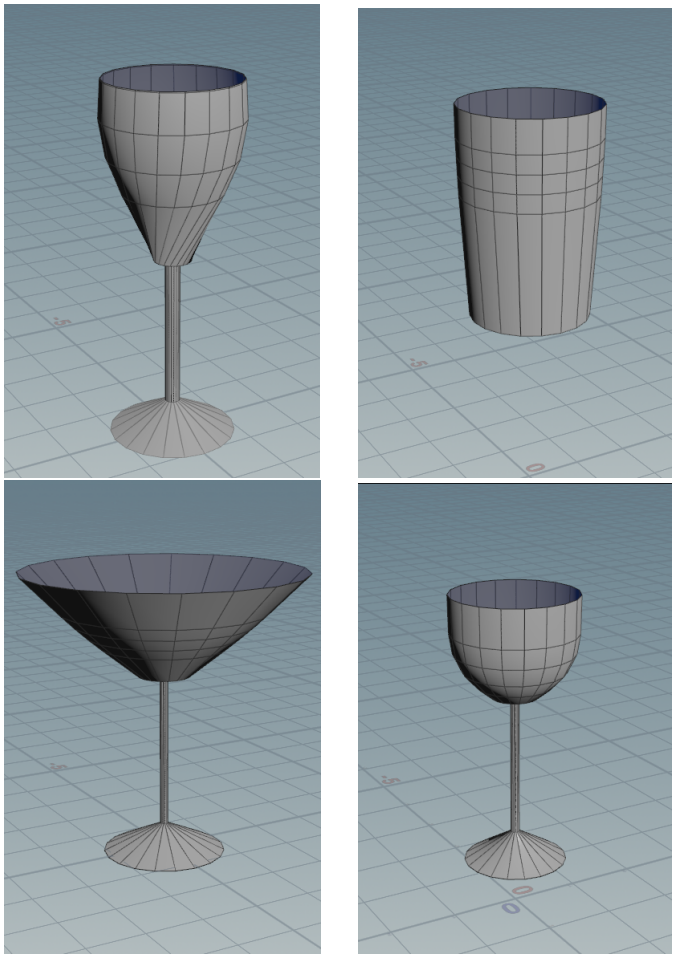


Figura 2.15: Bicchieri ottenibili dallo stesso modello

TV

Il modello della TV è composto da due cubi, uniti tra di loro per mezzo di un *Merge*. Il primo è modellato per ottenere lo schermo, mentre il secondo fa riferimento alla barra e al piede di appoggio. Le TV di ultima generazione possono anche essere appese al uro, pertanto il piedi con la propria barra possono essere nascosti con il nodo *Blast*. Questa opzione è stata anche aggiunta nell'asset: quando l'utente clicca sul bottone che permette di eliminare il piede, di conseguenza, anche le cartelle contenti i parametri relativi al piede e alla barra vengono disabilitate con la seguente istruzione:

```
1    { negate == 0 }
```

dove *negate* è il nome del bottone.

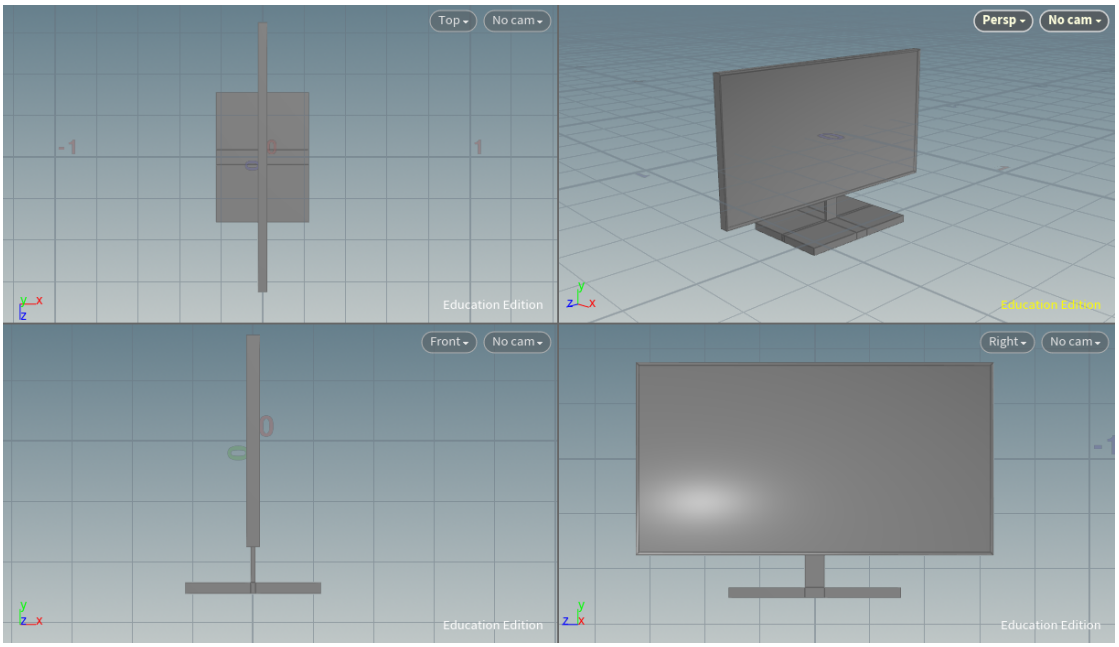


Figura 2.16: Modello 3D della TV

Parametri procedurali
Dimensione XYZ
Larghezza cornice
Larghezza della barra
Altezza della barra
Altezza del piede
Curvatura del piede
Larghezza del piede
Lunghezza del piede

Tabella 2.9: Parametri procedurali della TV

Mobile per la TV

Forse uno dei modelli più semplici, il mobile console è composto da un cubo che può essere moltiplicato del numero desiderato. L'incremento viene effettuato per mezzo del nodo *Copy stamp*, che aggiunge un nuovo elemento traslato in X di un fattore pari alla sua dimensione di scala X, ovvero:

```
1    ch("../transform1/sx")
```

Parametri procedurali
Scala
Estrusione
Bombatura
Numero elementi

Tabella 2.10: Parametri procedurali del mobile per la TV

La faccia anteriore del cubo è stata estrusa in avanti; anche questo parametro è gestibile dall'utente.

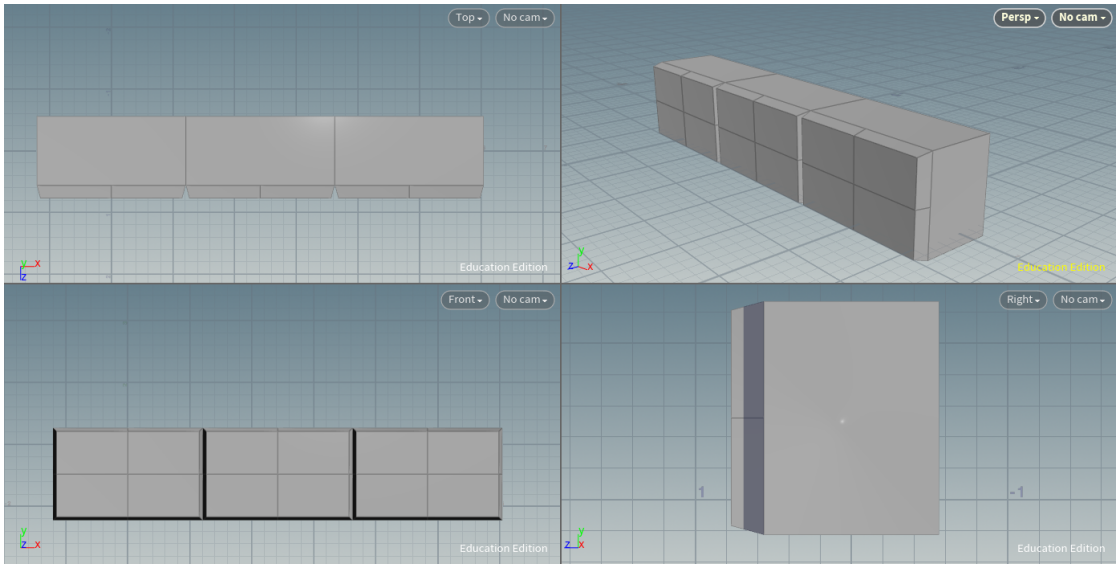


Figura 2.17: Modello 3D del mobile per la TV

Libri

Per riempire ancora di più la stanza, la libreria poteva ancora essere completata con l'inserimento di alcuni libri negli scaffali. Lo schema di nodi impiegato nella modellazione di ciò che vedete nell'immagine 2.18 sono è simile ai precedenti.

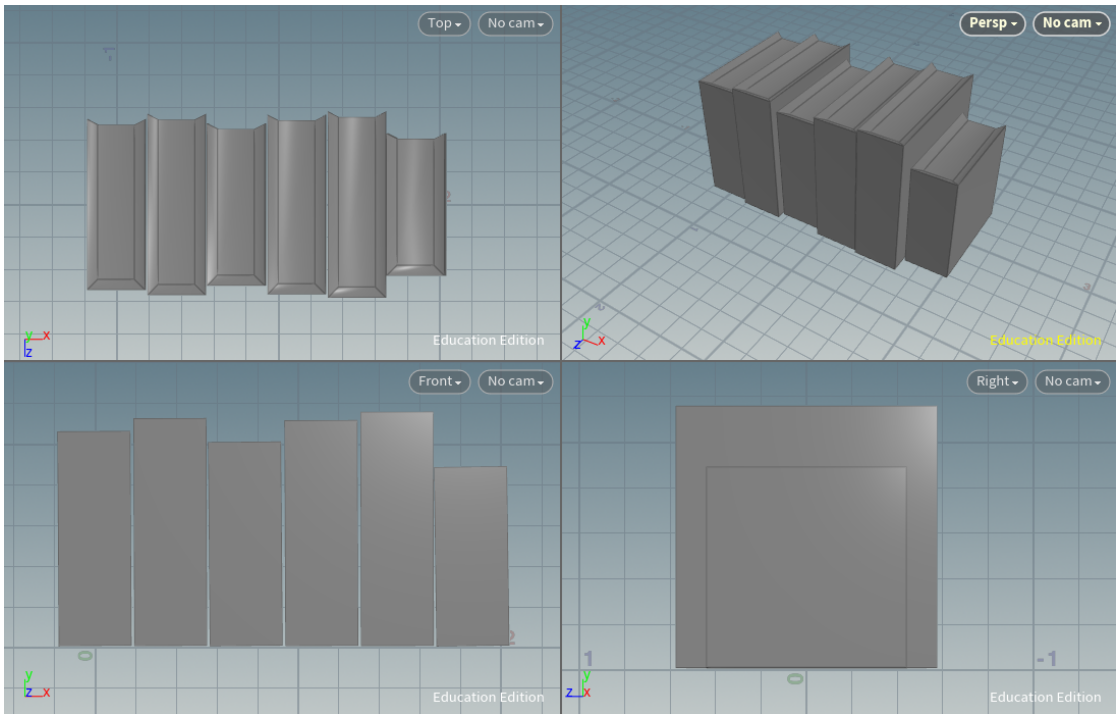


Figura 2.18: Modello 3D dei libri

Come prima cosa, bisogna modificare un cubo e dargli l'aspetto di un libro. Poi, attraverso un ciclo che copia n-volte la geometria che riceva in ingresso, si ottiene la fila di libri impilabili. La distribuzione nello spazio segue una linea orizzontale la cui lunghezza è data dalla dimensione del libro per il numero di libri voluti, meno un certo distanziamento:

```
1    ch("../transform1/sx")*ch("points")-0.3
```

Al fine di ottenere un ordine sparso, come per il caso delle bottiglie, anche qui è stato aggiunto un fattore di casualità, rispettivamente, nella rotazione lungo l'asse Z e nella dimensione in Y e Z:

```
fit01(rand(detail("../fb_metadata1/", iteration, 0)), 1, 0)
```

```
fit01(rand(detail("../fb_metadata1/", iteration, 0)), 0.8, 1.2)
```

Parametri procedurali
Numero libri

Tabella 2.11: Parametri procedurali dei libri

Stanza

Obiettivo della tesi era la modellazione di una stanza interamente procedurale. Adesso che gli oggetti ci sono tutti, manca solo il posto fisico.

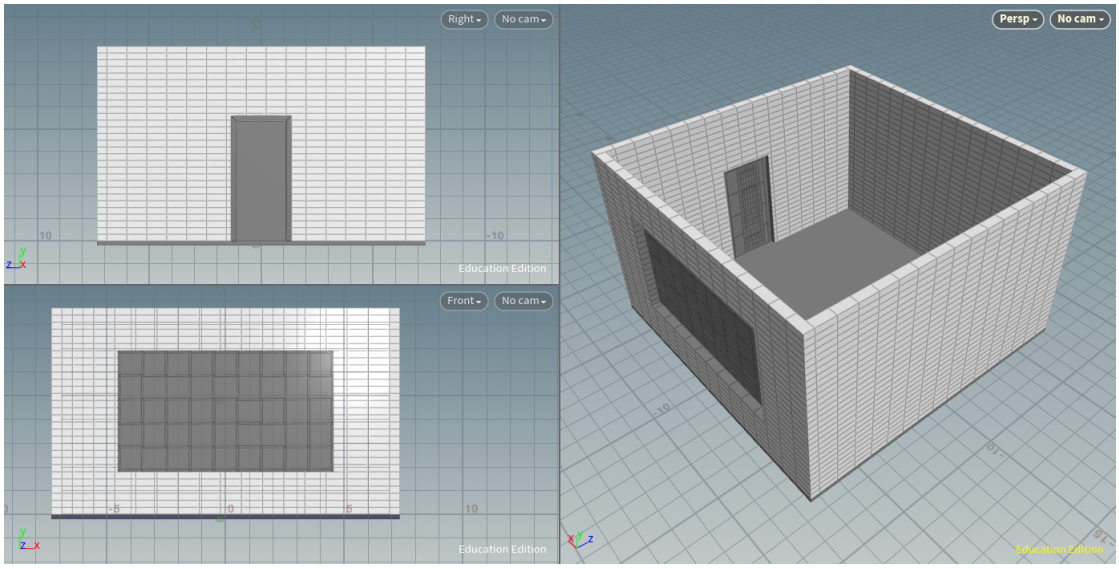


Figura 2.19: Modello 3D della stanza

Gli unici parametri procedurali riguardanti la camera sono la lunghezza e l'altezza,

come riporta la tabella 2.12. Il primo si ricava per mezzo di una linea orizzontale, lunga pari al numero di segmenti che compongono una fila di mattoncini per la dimensione di uno di questi:

```
ch("../length/segs") * ch("../box1/sizex")
```

Il secondo parametro, invece, è dato dal nodo *Copy and transform* che, appunto, copia la linea e la incolla di volta in volta traslata verso l'alto di quanto è la dimensione in Y del mattoncino. Il numero di copie totali detta l'altezza della stanza.
Abbiamo quindi creato un muro fatto di linee orizzontali. Per le tre pareti restanti basta applicare, in ordine:

1. una trasformazione in Z pari a

   ```
   ch("../line1/dist") * -0.5 - (ch("../box1/sizex") * 0.75)
   ```

   dove il primo termine è uguale all'origine in X della linea orizzontale, mentre il secondo permette di avere un margine di spazio alla fine della parete per evitare che i mattoncini si intersechino;

2. un *Copy and transform* ruotato di 90° lungo l'asse Y e con numero di copie pari a 4.

Infine, i mattoncini andranno ad occupare il loro posto attraverso il nodo *Copy to points* (vedi figura 2.20).

Per l'aggiunta del pavimento e del soffitto, è bene impiegare il nodo *Bound*, in quanto crea una riquadro di delimitazione attorno alla geometria di input, seguito da un *boolean*, che semplicemente sottrae la parte che si vuole visualizzare dal resto del contorno prima creato.

Finestra e porta hanno bisogno di un buco nelle mura per l'inserimento. Soprattutto la finestra, alla quale verrà comunque attribuito un materiale trasparente. Piuttosto che utilizzare i procedimenti già presentati, ovvero con un *Merge* o con una serie di trasformazioni ed estrusioni o con uno *Knife*, il metodo qui utilizzato è un altro ancora. Tutto ciò ci permette di osservare, inoltre, quanto vasto sia il mondo di Houdini nella modellazione e quanto il modellatore possa sbizzarrirsi e usare i nodi che più gli aggradano. Affronteremo singolarmente i due casi, con le rispettive fessure.

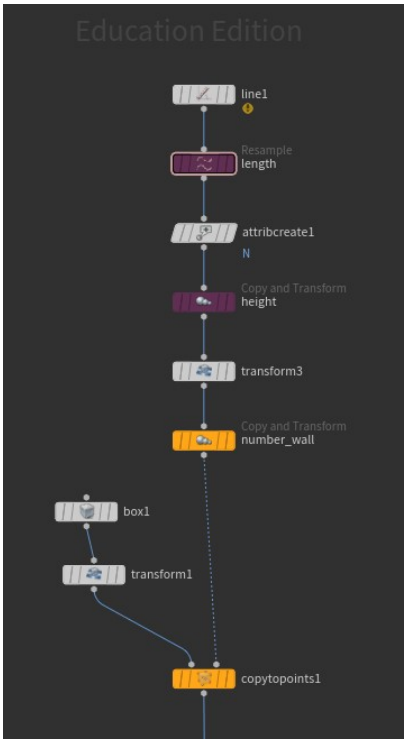


Figura 2.20: Gerarchia dei nodi per la modellazione delle pareti

- **Finestra**

  Dal momento che la finestra può essere allargata e allungata, per dare un senso più armonioso, la fessura è stata fatta al centro del muro. Utilizzare un cubo con primitiva una mesh, e non un poligono, consente di suddividere il *Bound* ricevuto in ingresso. La parete scelta, una volta essere stata divisa in quattro, avrà un solo vertice, perfettamente al centro, in quanto centro della croce. Usando un *Copy to points*, si va ad inserire in quella precisa posizione un quadrato che ha le dimensioni della finestra voluta.

  Gli ultimi due parametri procedurali presenti nella tabella 2.12 fanno riferimento alla finestra. Scegliendo il valore minimo in entrambi, cioè 1, si ottiene un piccolo quadrato; incrementando questo valore, aumenta anche il numero di quadratini. Si può dare, insomma, alla finestra una forma stretta e lunga o ancora schiacciata e larga, come più si vuole. Ad esempio nell'immagine 2.19, la lunghezza è pari a 9 e l'altezza a 5.

La modellazione vede l'unione di due linee, una orizzontale e una verticale, con lunghezze dettate rispettivamente da:

```
ch("points") - ch("../transform8/sx")

ch("points") - ch("../transform5/sy") + 0.11
```

Tranform8/sx e *tranform5/sy* si riferiscono alle dimensione dei parallelepipedi che fanno da cornice al vetro.
Per aumentare i quadrati all'aumentare del valore scelto dall'utente, è chiaro che vi è bisogno ancora una volta di fare un qualche copia-e-incolla. Entrambe le linee entrano infatti in due cicli for, separati, contenenti un *Copy stamp* ciascuno, dove viene incrementato di uno il valore scelto dall'utente con la seguente regola:

```
ch("../base/points") + 1
```

Il +1 è fondamentale in quanto il conteggio del numero di copie di default inizia da 0.

Adesso, se colleghiamo quanto appena costruito con la fessura apposita, tramite un *Copy to points*, il vertice in basso a sinistra della finestra si collocherà esattamente al centro della parete che avevamo scelto all'inizio. Per far combaciare i due elementi dobbiamo effettuare delle operazione di trasformazione. La finestra verrà traslata in:

- X:

```
        ch("../base/points") / 2 - (ch("../base/points") -
            0.5)
```

- Y:

```
        - ch("../altezza/points") / 2
```

- Z:

```
        - 0.5
```

Mentre l'apertura dovrà adattarsi all'altezza e alla lunghezza scelta:

– X:

```
ch("../base/points")
```

– Y:

```
ch("../altezza/points")
```

Infine, è stato aggiunto il vetro, creato a partire da un cubo e ridimensionato in funzione della finestra (vedi Appendice A).

Lo schema 2.21 aiuta a capire i collegamenti nella gerarchia.

- **Porta**
  Allo stesso modo della finestra, prima è stata creata l'apertura attraverso un *Bound*, opportunamente tagliato con dei *Poly Split*[3] e *Poly Cut*. Successivamente, il cubo modellato in funzione della porta è stato posizionato con un *Copy to points*. La sua scala in Z però deve essere legata proceduralmente al resto della struttura, nel modo seguente:

```
ch("../length/segs") / 15 * 0.9
```

dove 15 rappresenta il valore minimo accettabile per la lunghezza della casa.

La porta vera e propria è stata modellata a parte e intersecata al resto per mezzo di un booleano.

Prima dell'output finale, i vari elementi sono stati legati con un *Merge*.

[3]Del nodo, è stata selezionata la modalità *Edge loop*, in modo da conservare gli attributi dei vertici.

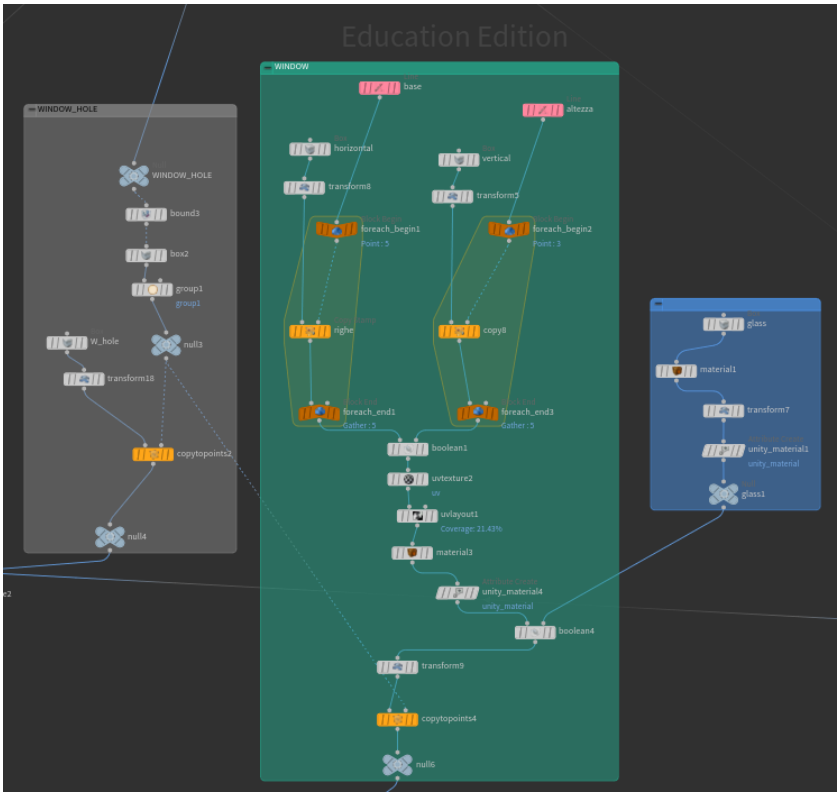


Figura 2.21: Gerarchia dei nodi per la modellazione della finestra

Parametri procedurali
Lunghezza
Altezza
Finestra - lunghezza
Finestra - altezza

Tabella 2.12: Parametri procedurali della stanza

2.2 Materiali

I materiali sono come degli asset racchiusi all'interno della rete VOP. Per assegnare un materiale ad un oggetto, basta trascinare lo shader sulla geometria del modello, oppure si può utilizzare il nodo **Metarial**, a livello SOP, e scegliere come target d'interesse solo alcune facce della mesh. Houdini mette a disposizione una palette di materiali già pronta, contenente le seguenti categorie:

Categoria	**Materiale**
Personaggi	pelle, capelli
Vestiti	velluto, pelle, cotone
Vetro	trasparente, colorato, ruvido
Industriali	gomma, plastica, porcellana, vernice, mattoni, calcestruzzo
Liquidi	liquido normale, oceano
Metalli	alluminio, argento, oro, acciaio, metallo colorato
Miscele	cera, cioccolato
Natura	marmo, roccia, argilla, trucioli di legno
Volumi	nuvole, fiamma, fumo, candela, palla di fuoco

Tabella 2.13: Material Palette

2.2.1 Come rendere procedurali i materiali

Data la natura altamente procedurale di Houdini, lo stesso principio della modellazione si può applicare al mondo dei materiali. A volte, insieme alle modifiche della mesh, è opportuno che anche il materiale applicato cambi di conseguenza. Supponiamo ad esempio di allungare la lunghezza della stanza. Se la texture è impostata come fissa, allora potrebbe tirarsi troppo da risultare sgranata. Oltre al più comune nodo Material, infatti, esistono diversi strumenti, molti dei quali presenti nello scaffale, per l'applicazione delle **texture** e dell'**UV map**[4].
In alcuni casi è bastato inserire anche un solo nodo nella rete della geometria, in altri, invece, la manipolazione ha richiesto qualche passaggio aggiuntivo. Di seguito verranno pertanto elencati quali nodi sono stati utilizzati e cosa comporta il loro

[4]Le texture sono delle trame, sotto forma di immagini 2D, da appiccicare al modello 3D grazie ad un'apposita mappatura in coordinate UV.

utilizzo.

- **UV Texture**: assegna le coordinate UV della trama all'oggetto 3D. A seconda della geometria su cui si sta lavorando, è opportuno scegliere la giusta proiezione. Ad esempio, per il modello del divano è stato utilizzato il tipo *Face*, poiché una copia della texture viene mappata su ogni faccia lungo la sua normale, orientandosi automaticamente senza ridimensionarsi. Per i piedi del tavolo la proiezione era invece cilindrica, indirizzata lungo l'asse y.
- **UV Layout**: racchiude le mappe UV all'interno di un'area ben specifica, chiamata *isola*. Poiché la mappatura avviene su un classico piano cartesiano in due dimensione, questa zona può essere facilmente modificata, in quanto soggetta a trasformazioni. Grandi superfici 3D avranno grandi isole UV e piccole superfici avranno piccole isole. Il nodo è stato usato nel modello del mobile per la TV, con l'impostazione che l'isola si adatti automaticamente alla superficie.
- **UV Unwrap**: a differenza dell'UV texture di tipo face, la mappatura risulta completamente piatta, quindi senza la minima distorsione, e non sovrapposta. L'UV unwrap è stato utilizzato per il piano del tavolo perché ,in seguito ad un ridimensionamento in X e in Z, la texture si adatta in automatico.
- **UV Transform**: permette di trasformare le coordinate della trama sulla geometria di origine. Esattamente come nel caso della modellazione, anche in questo caso si possono collegare i parametri di traslazione, rotazione e di scala ad altri parametri della stessa rete, a seconda delle proprie esigenze.

2.3 Digital Assets

Secondo la pipeline di produzione presentata in questa tesi, dopo le fasi di modellazione e texturing, si entra in quella di *Asset e UI*. In Houdini, infatti, le reti, con i rispettivi nodi, possono essere ulteriormente raggruppate. Questo nuovo insieme prende il nome di Digital Asset e viene archiviato in un *digital asset library file*, con estensione di tipo **.hda** (Houdini Digital Assets). Si tratta di un passaggio molto interessante, soprattutto se si vuole rendere la rete di sola lettura o se si vogliono mettere in evidenza, nell'interfaccia, i parametri procedurali che gli artisti possono modificare. Infatti, basta impacchettare il flusso di lavoro ed inserire, nelle proprietà dell'asset, il range dei valori per ciascuno dei parametri, come mostra la figura 2.22. I parametri inseriti nell'asset possono essere organizzati in modo da creare un'interfaccia utente pulita e funzionale, per mezzo di cartelle e di separatori, o dinamica, legando la loro presenza ad altri parametri. Ad un'interfaccia curata corrisponde, quindi, maggiore semplicità per l'utente nel suo utilizzo.

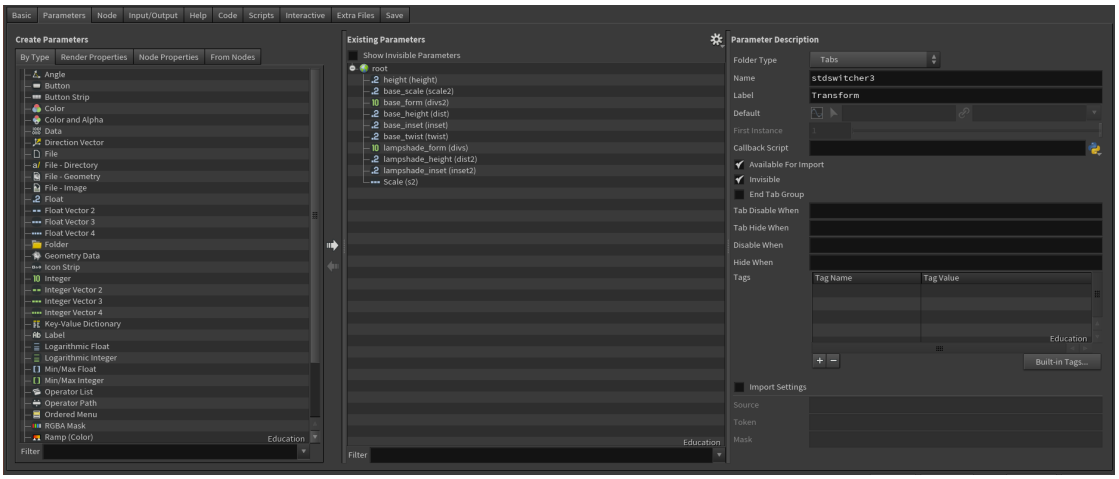


Figura 2.22: Edit Operator Type Properties

Per inserire un parametro dalla rete di Houdini, basta trascinarlo nell'asset e le proprietà padre-figlio rimangono inalterate; in alternativa, i parametri si possono creare da zero. Questo è molto utile soprattutto quando si vuole legare al parametro uno script o un qualche riferimento particolare. I tipi di parametri supportati, visibili nella parte sinistra dell'immagine, sono:

- floating point
- integer
- stringhe

- bottoni
- colori
- UV
- file
- label

In previsione di un futuro inserimento di un avatar digitale, un obiettivo della tesi era la possibilità di permettere all'ambiente di cambiare velocemente aspetto in modo casuale. Poiché la modellazione procedurale permette di creare oggetti che sono in grado di modificarsi in seguito alla modifica di pochi elementi, bisognava inserire un parametro che andasse a modificare tutti gli altri per avere diversi aspetti dell'ambiente il più velocemente possibile. Nell'asset è stato quindi aggiunto un elemento di tipo *float*, al quale è stato collegato uno script che permette di regolare tutti gli altri parametri dello stesso asset. Precisamente, per ogni parametro contenuto nell'asset, viene pescato in modo random un valore all'interno del suo range (*bounds[]* nello script). In questo modo, all'utente basterà utilizzare un solo nodo per cambiare la forma all'oggetto, piuttosto che andare a modificare tutti i parametri disponibili. [17]

```
    for parm_name in parm_names:
        parm = node.parm(parm_name)
        value = hou.hscriptExpression(
            "fit(rand(%s),0,1,%s,%s)" % (
                hash(parm_name) % 1000000 + seed,
                bounds[0], bounds[1]))
        if parm.parmTemplate().dataType() == hou.parmData.Int:
            value = int(value)
        parm.set(value)
```

Per lo script completo vedere Appendice A.2.

Creare i Digital Assets è stato essenziale ai fini dell'interazione tra Unity e gli oggetti 3D modellati su Houdini. Dopo che l'asset è stato inserito nel progetto di Unity, l'interfaccia creata nelle proprietà dell'asset è visibile su Unity, in un'apposita finestra. Per questo motivo l'UI va creata in modo accurato, in quanto l'utente, o il programmatore, si interfacciano con questa per apportare modifiche al modello. L'integrazione dei due software è stata resa possibile grazie al plug-in *Houdini Engine for Unity*, discusso al paragrafo 3.2.

Capitolo 3

Proceduralità in real-time

3.1 Il motore grafico Unity

Per la seconda parte del progetto di tesi, è stato utilizzato il *game engine* di Unity[1] principalmente per due motivi:

- consente di avere una resa grafica ad alta definizione, utile per l'ultimo step della pipeline di produzione
- permette l'integrazione con il software Houdini, per mezzo di un plug-in

In quanto finalizzato allo sviluppo di giochi e applicazioni di realtà virtuale, lo *scripting* all'interno di Unity è un elemento chiave. Tramite poche righe di codice è possibile, inoltre, rendere procedurale l'ambiente in cui ci si trova, come ad esempio un livello di gioco. Per capire in che modo fosse possibile realizzare spazi ed oggetti in modo procedurale su Unity, è stato creato uno script che genera in modo casuale una città, mostrata nell'immagine 3.1. I palazzi, le strade e gli alberi sono stati importati come dei semplici oggetti 3D - non procedurali - ed inseriti in appositi array[2] per poter essere istanziati una volta fatto partire il generatore. Il modo più semplice per creare situazioni casuali, come si evince anche dal capitolo precedente, è l'aggiunta di rumore. In questo caso, è stato utilizzato l'algoritmo di Perlin[3] (vedere appendice B.1 per ulteriori informazioni) che, grazie all'andamento graduale

[1] Il software è supportato da diversi sistemi operativi, ovvero Microsoft Windows, Linux e macOS, e può essere lanciato anche su PlayStation, Nintendo e Xbox

[2] In informatica, un array è un insieme indicizzato di elementi appartenenti allo stesso tipo.

[3] Sviluppato da Ken Perlin nel 1983, è un tipo di rumore sfumato e quindi usato in tecniche di computer grafica per le texture procedurali.

e non completamente casuale dei suoi valori, ha permesso di insediare certi edifici in quartieri piuttosto che in altri. [18, 19]

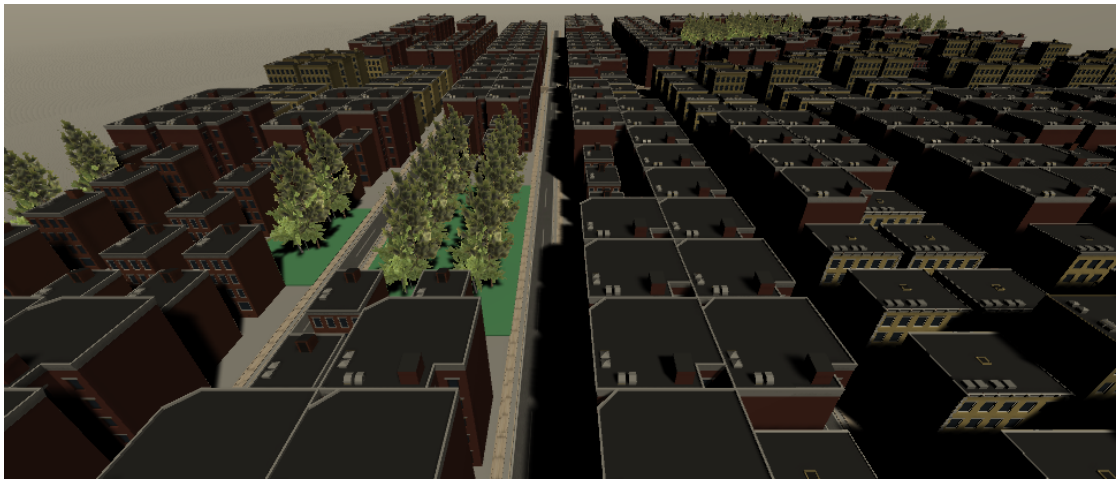


Figura 3.1: Generatore procedurale di una città

Nonostante questa premessa, è comunque sconsigliato usare esclusivamente Unity per la produzione procedurale dal momento che questo è un motore grafico e non un software di modellazione 3D, come lo è invece Houdini.

Versione	2019.3.11.f1
Template	HDRP

Tabella 3.1: Versione di Unity utilizzata

Come funziona Unity

Come nel caso precedente (paragrafo 2), parleremo brevemente dell'interfaccia di Unity. I modelli 3D, i materiali, gli script e tutto ciò che può servire al progetto vengono salvati in apposite cartelle visibili nella finestra del **progetto**. Da qui, possono essere trascinati nella **vista scena** o nella **gerarchia**, per diventare effettivamente oggetti con cui lavorare, o nell'**inspector** se rappresentano attributi aggiuntivi, come ad esempio i materiali. Dalla finestra dell'inspector, infatti, si possono gestire i vari oggetti, sia in termini di trasformazioni nello spazio che di interazione. Infine, cliccando su *play* dalla **barra degli strumenti**, quello da noi

creato prenderà vita nella **vista gioco**.[20]

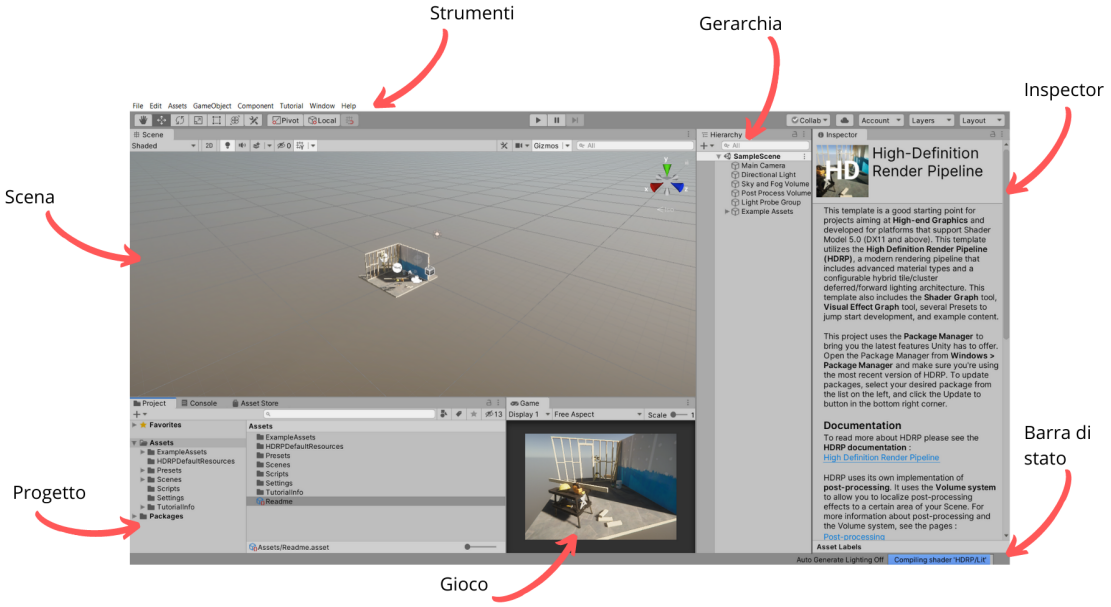


Figura 3.2: Interfaccia utente di Unity

3.1.1 High Definition Render Pipeline

Dall'inspector della figura 3.2, si nota che per il progetto è stato impiegato il template High Definition Render Pipeline (HDRP). Si tratta di una **pipeline di rendering**[4] creata appositamente per le piattaforme compatibili con i programmi che girano su schede grafiche, cioè i *Compute Shader*, che utilizza le leggi della fisica per dare la giusta illuminazione all'ambiente.

[4]Una pipeline di rendering esegue una serie di operazioni sul contenuto di un file Scena e ne permette la visualizzazione su schermo. Ciascuna pipeline di rendering ha prestazioni diverse che si adattano a certa applicazioni piuttosto che altre. É importante, quindi, scegliere quale pipeline utilizzare nel proprio contesto per non avere problemi di compatibilità in futuro.

Rispetto alle vecchie pipeline, l'HDRP apporta delle interessanti novità nei seguenti ambiti:

- **Materiali**: grazie ai *Lit Shader*, i materiali sono più realistici. Giocando con le opzioni che vengono messe a disposizioni, è possibile ottenere materiali che non subiscono alcun effetto dalla luce a cui sono esposti o materiali che trasmettono sensazione di cotone o seta.
- **Illuminazione**: le luci seguono le dimensioni fisiche e ciò permette di ottenere un certo naturalismo. Per la riflessione e la rifrazione, l'HDRP utilizza la profondità dello schermo in base ai movimenti della camera, grazie ai *reflection probe*. Ognuna di questa sonde memorizza una vista dell'ambiente e le proprietà dei materiali circostanti, per produrre riflessi in funzione dell'angolo di vista. Per quanto riguarda, invece, le ombre, queste vengono calcolate principalmente attraverso tre metodi:
  - **Shadow cascades**: tiene in considerazione la luce direzionale e la distanza dalla camera.
  - **Contact Shadows**: utilizza un buffer di profondità per catturare anche i dettagli più piccoli.
  - **Micro Shadows**: prende in considerazione la normal map e l'ambient occlusion map.

3.2 Houdini Engine

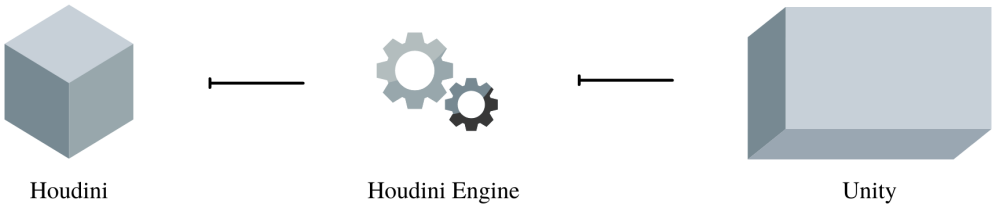


Figura 3.3: Integrazione Houdini Engine

Houdini Engine for Unity è il plug-in che permette l'integrazione tra Houdini e Unity, rendendo la geometria creata con il primo software compatibile con il secondo, per mezzo degli HDA (discussi al paragrafo 2.3). Dopo essere stato installato correttamente, come prima cosa, il plug-in apre automaticamente una sessione che mette in comunicazione i due software, attraverso una libreria client, HARC, ed un server, HARS. In ogni caso, se si è interessati, la sessione può essere configurata dalla finestra *Houdini Engine Settings*. Le sessioni sono di due tipi: pipe[5] e socket. [22]

- **Sessione pipe**: è la più semplice comunicazione tra due processi, nonché quella consigliata
- **Sessione socket**: utilizza una socket TCP[6] per la comunicazione

Per importare un oggetto 3D modellato su Houdini, basta trascinarlo all'interno del progetto di Unity affinché il plug-in crei una **risorsa Houdini**. Il *Game Object* generato dall'asset è organizzato secondo una gerarchia ben precisa. A fare da padre è il **Root Gameobject**, che permette di visualizzare i dati nell'Inspector grazie allo script *HEU_HoudiniAssetRoot*. Per quanto riguarda i figli:

1. **HDA_Data** contiene lo script principale per le interazioni Houdini-Unity, ovvero l'*HEU_HoudiniAsset*,

[5]La comunicazione pipe è un tipo di comunicazione diretta: ogni pipe client comunica con il pipe server. Supporta esclusivamente processi che condividono i descrittori di file. [21]

[6]Tipicamente, una socket TCP mette in comunicazione due programmi, lato client e lato server, attraverso una connessione TCP. In telecomunicazioni, il TCP (Transmission Control Protocol) è un protocollo orientato alla connessione, in quanto, prima dell'effettivo scambio d'informazioni, i due processi si scambiano dei dati preliminari; in gergo si dice che effettuano l'*handshake*. [23]

2. **Output GameObjects** permette la visualizzazione della mesh, materiali e collider.

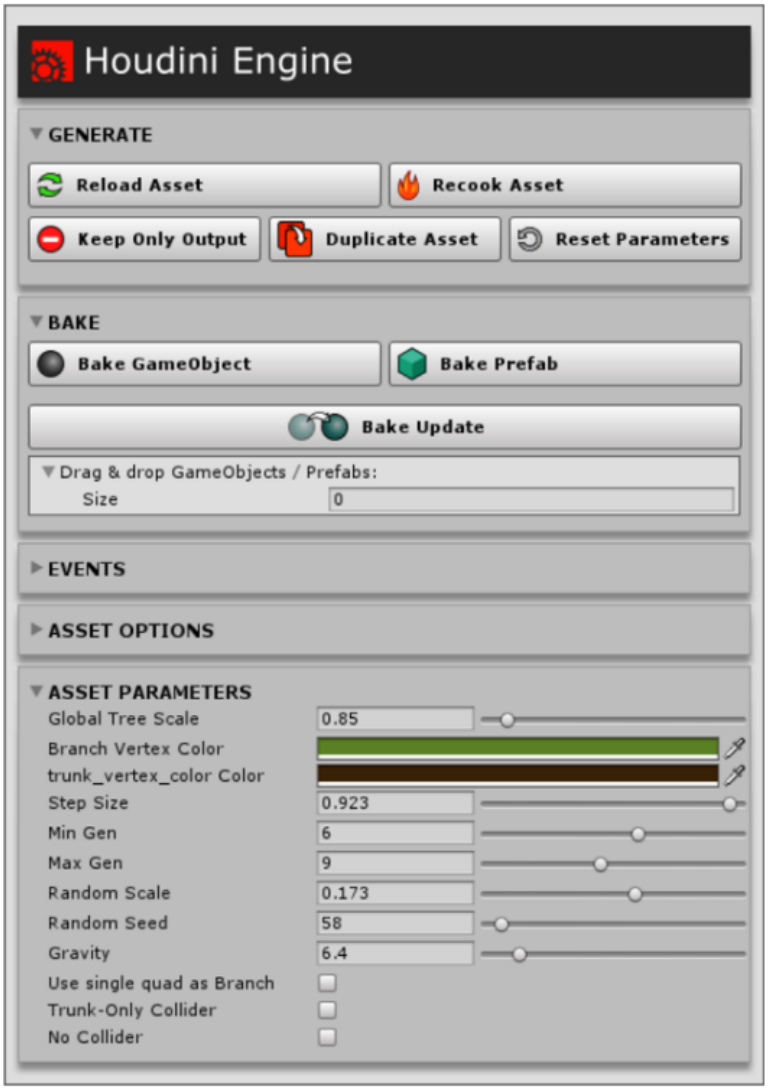


Figura 3.4: Interfaccia Asset Inspector

Una volta importato il file HDA, i parametri contenuti nell'asset sono visibili su Unity e modificabili, o tramite script o tramite l'apposita interfaccia di editor (*Asset Inspector*). Come si nota in figura 3.4, la finestra dell'Inspector può essere suddivisa in macro-blocchi. La prima parte riguarda il campo **Generate**, utile in particolare per la rigenerazione della mesh (*Recook Asset*), l'aggiornamento del modello 3D

(*Reload Asset*) e per fare copia e incolla (*Duplicate Asset*). Nella seconda sezione troviamo gli elementi per effettuare il **Bake**, cioè per creare delle copie dell'Output GameObject o per aggiornare il prefabbricato. Gli **Events** permettono di inserire callback e script personali, mentre gli **Assept Options** permettono di controllare la *cottura* della risorsa o di alcune parti di questa, come le normali o i materiali. In fondo si trovano invece i **Parametri** del file HDA, modificabili a proprio piacimento. [24]

Il plug-in, quindi, può anche essere usato nel proprio codice e non solo tramite UI, ma è bene innanzitutto capire come sono organizzati i livelli d'interfaccia della programmazione.

- **Asset layer**: viene usato per interagire con i parametri dell'asset o per effettuare delle semplici operazioni, come quella di *duplicate*. Al suo interno si trovano le classi *HEU_HoudiniAssetRoot*, *HEU_HoudiniAssetRoot* ed *HEU_ParameterUtility*.
- **Utility layer**: di medio-livello, si utilizza per lavorare con nodi, reti, mesh, e interagire con la sessione dell'engine.
- **HAPI layer**: permette di interagire, a basso livello, con l'API per ottenere un maggiore controllo del plug-in.

Nel lavoro di tesi, ci si è interfacciati principalmente con l'Asset layer, in quanto le funzionalità che sono state implementate richiedevano una manipolazione dei parametri procedurali dell'asset.

3.2.1 Funzionalità implementate

Sono stati costruiti due programmi, per vedere come rispondesse Unity al plug-in. Il primo permette di cambiare in modo casuale la posizione dell'oggetto selezionato, mentre il secondo mostra come poter risolvere un problema riscontrato su Houdini. Queste funzionalità non sono comunque complete, in quanto obiettivo della tesi è l'analisi della pipeline e non lo sviluppo su game engine.

Posizione random

Per approfondire e capire in che modo fosse possibile interagire con i parametri procedurali tramite gli strumenti di scripting offerti da Unity e dal plug-in, è stato realizzato un bottone che permette di spostare l'oggetto selezionato in una nuova posizione casuale all'interno della stanza in cui si trova. Il bottone in questione è

inserito in una finestra di editor, creata ad hoc ed ospitata dal menu dell'*Houdini Engine Settings*. Poiché il presupposto della tesi è la creazione di una stanza arredata casualmente per dimostrare le potenzialità della pipeline di produzione scelta, all'utente viene comunque data la possibilità di spostare i mobili, in due modi: o direttamente, usando le trasformazioni di traslazione di Unity, o in modo randomico tramite l'apposito bottone.
Come game engine, Unity permette di effettuare delle simulazioni basate sulle leggi fisiche, grazie ad un proprio motore fisico integrato. Al fine di evitare collisioni con gli altri oggetti presenti, è stato utilizzato il pacchetto **Unity Physics**, che rappresenta un ottimo sistema di dinamica del corpo rigido. In particolare, il metodo

```
1    Physics.OverlapSphere(Vector3 position, float radius)
```

racchiude l'oggetto in questione dentro una sfera e restituisce una lista di tutti gli oggetti con cui entra in contatto. Come parametri accetta la posizione esatta del centro dell'oggetto, in XYZ, e il raggio che deve avere la sfera per inglobare l'oggetto. Al raggio e alla posizione si può risalire tramite il metodo *Render.bounds*. Come prima cosa, bisogna accedere al componente *renderer* del game object, o meglio, all'output game object. Houdini permette di aggiungere alla geometria un *collider*, sia in renderer che senza, sia all'interno di scatole che di sfere. Il nodo in questione è un gruppo dal nome **collision_geo** e può essere abilitato anche dall'asset tramite un apposito bottone. (Appendice B.2)

Quella appena presentata è senza dubbio la strada più veloce, nonché quella più efficiente e precisa. Scopo della tesi è trovare vantaggi e svantaggi della pipeline di produzione che parte con la modellazione in Houdini e finisce con i testing dei modelli procedurali in ambiente real-time in Unity. Quindi per capire quale metodo fosse migliore da utilizzare per il recupero della posizione e del raggio, sono stati fatti degli altri esperimenti. Alla posizione si accede, comunemente, con il metodo *Transform.position*, seppure essere meno preciso del precedente. Senza un collider, a questo punto, il raggio coincide con la metà della dimensione dell'oggetto. A seconda del contenuto del file HDA, questa dimensione potrebbe trovarsi tra i parametri dell'asset. Grazie all'**asset layer**, è possibile interrogare e modificare i parametri dell'asset, attraverso la classe *HEU_ParameterUtility*. In particolare il metodo

```
1    GetFloat(HEU_HoudiniAsset asset, string paramName,
2             out float outValue)
```

restituisce il valore del parametro che gli viene passato come argomento.

Come si nota dal codice, il primo parametro in input riguarda l'asset del game object che vogliamo utilizzare. Per passare da un game object all'asset di riferimento bisogna prima inserire questa riga:

```
1    HEU_HoudiniAsset curAsset = curGameObject.GetComponent<
    HEU_HoudiniAssetRoot>() != null ? curGameObject.GetComponent<
    HEU_HoudiniAssetRoot>().curAsset : null;
```

Fila di libri

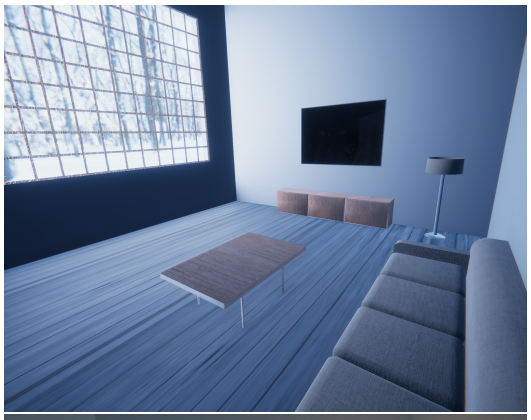
Nel capitolo precedente sono stati presentati i vari oggetti e come sono stati modellati con Houdini. In particolare, nel caso dei libri, sono nate delle criticità in fase di texturing. L'idea era quella di creare una fila di libri diversi tra loro, per essere inserita poi nella libreria. Ogni copia del libro, quindi, deve quindi avere una copertina diversa dalle altre copie. La texture utilizzata è composta da più copertine, distribuite in modo omogeneo. A causa del ciclo impiegato per la creazione delle copie, l'UV-map di ciascuna copia si accavalla su quella precedente. Sono state tentate diverse strada, come l'*UV transform* o lo spostamento dei nodi delle texture sia prima che dopo il ciclo . Si è pensato allora di ovviare al problema utilizzando gli strumenti di scripting di Unity, per mezzo del plug-in. L'algoritmo è lo stesso di quello pensato per Houdini: ad ogni iterazione di un ciclo *for* viene creata una copia, posizionata accanto alla precedente, e con una propria copertina. Su Unity, è possibile spostare l'UV, ma bisogna anche cambiare materiale ad ogni copia, perché l'UV rimane collegato al materiale di appartenenza. Per accedere alle risorse del progetto, si può utilizzare la classe *Asset Database*, che memorizza i dati dal file sorgente in modo da renderli utilizzabili dalle applicazioni. Il database creato contiene, quindi, il materiale e la rispettiva texture. La copia viene creata richiamando il metodo

```
1    public GameObject DuplicateAsset();
```

appartenente alla classe *HEU_HoudiniAsset*. Poiché il copia e incolla può non funzionare perfettamente in presenza di file HDA, l'Houdini Engine mette a disposizione sia la funzione che il bottone, presente nell'Inspector, per effettuare i duplicati.

3.3 Composizioni della stanza

Grazie al parametro che permette di raggiungere, potenzialmente, un numero infinito di randomicità (vedi Appendice A.2), i diversi livelli di composizione della stanza sono veramente tanti, dalla più ordinata, alla più caotica. Le immagini seguenti mostrano come la stanza può cambiare aspetto, seguito alla modifica di pochi modelli, quali divano, lampada, tavolo e stanza stessa.



Capitolo 4

Risultati ottenuti

Adesso che la pipeline per la generazione di modelli procedurali utilizzati in un ambiente real-time è stata ampiamente discussa, è il momento di analizzare quanto effettivamente sia efficiente, in termini di vantaggi e svantaggi.

4.1 Vantaggi

Modellazione procedurale con Houdini

La modellazione procedurale presenta di per se' una serie di vantaggi, già discussi al paragrafo 1.1.1. L'utilizzo di Houdini come software per la modellazione è sicuramente un ottimo punto di partenza. L'organizzazione di reti che contengono nodi permette di arrivare allo stesso modello in svariati modi. Ciò comporta maggiore flessibilità e mette il modellatore anche meno esperto completamente a suo agio, permettendogli di usare gli strumenti che più ritiene adatti.

Creazione UI

La creazione di asset e della UI è una fase cruciale della pipeline. Tutto quello che viene qui definito, viene dato poi in mano all'utente. Diventa essenziale, pertanto, che l'interfaccia utente sia quanto più comprensibile ed immediata. In questo senso, Houdini ci viene sicuramente incontro. Il poter raggruppare i parametri in cartelle e sottocartelle, dividerli grazie a degli appositi separatori o nasconderli quando viene eseguita un'azione su un qualche parametro in particolare, non è da sottovalutare.

Script di esempio nell'Houdini Engine

Capire in che modo interagiscono i due software non è immediato. Gli script di esempio sono abbastanza d'aiuto e offrono un quadro completo di tutte le

cose che si possono fare per mezzo del plug-in. La manipolazione dei parametri dell'asset da Unity e i metodi messi a disposizione dalle classi del plug-in rispondono perfettamente a quello che era l'obiettivo della tesi.

Livello di casualità

Fin da subito è stato posto l'accento su quanto fosse efficiente avere più versioni dello stesso oggetto a partire da un solo modello. Se però l'utente o il game designer devono gestire i singoli parametri procedurali, le operazioni da fare possono anche diventare onerose. L'aver inserito un nodo nell'asset stesso, che sia in grado di cambiare tutti gli altri paramatri in modo completamente random, permette di raggiungere un livello ancora più completo di casualità. In questo modo, basta modificare un solo parametro per passare da un oggetto ad un altro.

4.2 Svantaggi

Texture

Dal paragrafo 2.2.1 si evince che la proceduralità nei materiali va altrettanto bene come nella modellazione. In realtà, sono state riscontrate delle criticità quando nella rete SOP veniva utilizzato un ciclo for-each per aumentare il numero di copie con il nodo *Copy to points*. Nel paragrafo 2.1.1 sono stati presentati diversi modi di fare copia e incolla, ma il modello che più ha creato problemi è quello dei libri. L'idea iniziale era quella di avere una texture contenente diverse copertine di libri, distribuite in modo omogeneo; ciascuna copia avrebbe dovuto avere una copertina. Nel ciclo, però, l'UV-map di ogni copia si accavallava su quella precedente. Sono state tentate diverse strada, come l'*UV transform* o lo spostamento dei nodi delle texture sia prima che dopo il ciclo.

Colori e materiali in HDRP

- **Colori**
  Prima di testare i propri file HDA in Unity, è stato utilizzato l'asset di esempio, contenente un modello 3D di una palma, Tra i suoi parametri (per i parametri completi vedere la figura 3.4) ce ne sono due che permettono di cambiare il colore di foglie e tronco. Houdini, in quanto software di modellazione, ha nodi che riguardano i materiali, le texture ed anche il colore. Quest'ultimo si può quindi inserire nell'asset e diventare, pertanto, procedurale, ma non risulta compatibile con l'HDRP. Il problema non si presenta, invece, se si lavora con pipeline di rendering non ad alta definizione.

- **Materiali**
  Come per i colori, anche i materiali non sono, immediatamente, compatibili con l'HDRP. Per ovviare al problema, in Houdini esiste un nodo, *Unity_material*, a cui si può passare la stringa del path del materiale in Unity. Ad esempio, supponiamo di voler utilizzare il modello del tavolo. Prima di creare l'asset, dovrò inserire nella rete SOP il nodo *Unity_material* e, nell'apposito spazio nell'editor dei parametri, scrivere *"Assets/Materials/Wood.mat"* se il mio materiale si trova in quella cartella. Con geometrie relativamente semplici, con questo approccio si risolve il problema, ma se la mesh è ben articolata, allora bisogna fare attenzione a dove posizionare il nodo nella rete. In particolare, nei modelli che sono stati realizzati in questa tesi, il nodo *Unity_material* è stato posto in prossimità dei nodi *Material* o *Merge*.

Capitolo 5

Conclusioni

Obiettivo della tesi era l'analisi di una pipeline per la produzione di una stanza arredata proceduralmente, per mezzo del software Houdini, e posta in ambiente real-time, ovvero in Unity, per dimostrare le potenzialità della modellazione procedurale.

Houdini si è dimostrato un ottimo software per la modellazione procedurale. Gli strumenti messi a disposizioni sono veramente tanti: si passa dai più semplici, quali traslazioni ed estrusioni, a nodi che mettono in pratica la logica base dell'informatica, come cicli *for-each* e *while*. Il poter intervenire su alcuni parametri tramite linguaggio di programmazione apre poi le strade ad innumerevoli modi di creare il proprio oggetto.

Il plug-in **Houdini Engine for Unity**, seppure a volte bisogna intervenire manualmente per riconnettere il client al server, rappresenta un ottimo mezzo per l'integrazione con Unity. Essenziale è stata la fase di creazione di asset, che ha permesso di definire in modo dettagliato quali parametri sarebbero stati presenti nell'interfaccia, ai fini di una buona *user experience*.

L'utilizzo di modelli parametrici su un motore grafico, quale **Unity**, permette di creare mondi e livelli di gioco interamente procedurali. Ne consegue una serie di vantaggi, sia lato artista/programmatore, sia lato giocatore. I primi in quanto hanno a che fare con lo stesso oggetto 3D, il secondo perché vive un'esperienza di gioco completa, di volta in volta diversa dalla precedente.

5.1 Sviluppi futuri

In questo progetto è stata realizzata una sola camera, ma in previsione di un lavoro più ampio si può sicuramente pensare di ingrandirla o di creare una casa con più stanze.
I mobili contenuti al suo interno possono essere, come abbiamo visto, riposizionati in modo casuale; lo stesso concetto si potrebbe applicare anche agli oggetti che non poggiano direttamente sul pavimento, come tazze e quadri.
Attualmente i paramatri vengono modificati in editor mode, si potrebbe estendere la modifica anche in modalità di gioco, per mezzo di un avatar digitale, per una resa ancora maggiore ed evidente in real-time.

Appendice A

Houdini

A.1 Modello 3D: stanza

Window's glass translate:

```
ch("../base/points") / 2 - 0.5
ch("../altezza/points") / 2
0
```

Window's glass scale:

```
ch("../base/dist") + 1
ch("../altezza/points")
ch("../transform8/sz") - 0.02
```

A.2 Parameter Randomize

```
    def randomize(node):
    list_a = ["height", "length"]
    bounds_a = [0.75, 1.75]

    set_parms(node, list_a, bounds_a)

    def set_parms(node, parm_names, bounds):
    seed = node.evalParm("seed")
    for parm_name in parm_names:
        parm = node.parm(parm_name)
        value = hou.hscriptExpression(
            "fit(rand(%s),0,1,%s,%s)" % (
                hash(parm_name) % 1000000 + seed,
                bounds[0], bounds[1]))
        if parm.parmTemplate().dataType() == hou.parmData.Int:
            value = int(value)
        parm.set(value)
```

Appendice B

Unity

B.1 Perlin noise for procedural city

```
1     int[,] mapgrid = new int[mapWidth, mapHeight];
2     float seed = Random.Range(0, 100);
3     for (int h = 0; h < mapHeight; h++) {
4         for (int w = 0; w < mapWidth; w++) {
5             mapgrid[w, h] = (int)(Mathf.PerlinNoise(w / 10.0f + seed,
6             h / 10.0f+seed) * 10);
7         }
8     }
```

B.2 Collision

```
1     Renderer renderer = curObj.GetComponentInChildren<Renderer>();
2     Collider[] hitColliders = Physics.OverlapSphere(
3                               renderer.bounds.center,
4                               renderer.bounds.extents.magnitude);
5     foreach (var hitCollider in hitColliders) {
6         if (hitCollider.gameObject.name != curObj.name) {
7             if (hitCollider.gameObject.name != house_name) {
8                 return true;
9             }
10            else if (hitCollider.gameObject.name !=  house_name
11                    && hitCollider.gameObject.name != name)
12                return true;
13            else
14                return false;
15        }
16    }
```

Bibliografia

[1] David S. Ebert, F. Kenton Musgrave, Darwyn Peachey, Ken Perlin, John C. Hart e Steven Worley. *Texturing Modeling: A Procedural Approach*. 3ª ed. Morgan Kaufmann, 2003 (cit. a p. 1).

[2] Antonio Francesco Bazzoni. *Studio e implementazione di metodologie procedurali per la rappresentazione tridimensionale di edifici urbani ad alta complessità*. 2005. URL: https://etd.adm.unipi.it/theses/available/etd-01172006-123143/unrestricted/Tesi1.PDF (cit. a p. 1).

[3] Bruce Eckel. *Thinking in Java*. 4ª ed. Vol. 1 - I fondamenti. Pearson, 2006 (cit. a p. 1).

[4] Scuola Superiore Sant'Anna. *Procedural modelling*. 2007. URL: http://percro.sssup.it/~marcello/didattica/AA2007/15_ProceduralModeling.pdf (cit. a p. 2).

[5] *Modellazione procedurale: una breve introduzione*. Nov. 2020. URL: https://www.datagen.tech/procedural-modeling/ (cit. a p. 3).

[6] *Vantaggi della generazione procedurale*. URL: https://subscription.packtpub.com/book/game_development/9781785886713/1/ch01lvl1sec13/benefits-of-procedural-generation (cit. a p. 4).

[7] Matteo Bittanti. URL: https://www.mattscape.com/italian.html (cit. a p. 5).

[8] Matteo Bittanti. *Schermi interattivi. Il cinema nei videogiochi*. Booklet Milano, 2008, p. 214 (cit. a p. 5).

[9] Gillian Smith. «An Analog History of Procedural Content Generation». In: *Foundations of Digital Games*. 360 Huntington Ave, 100 ME, Boston, MA 02115, USA, 2015 (cit. a p. 5).

[10] Tom Hatfield. *Rise Of The Roguelikes: A Genre Evolves*. 2013. URL: http://pc.gamespy.com/pc/ftl-faster-than-light/1227287p1.html (cit. a p. 5).

[11] R. Hunter Gough. *I Hate Roguelikes, And So Should You!* 2020. URL: https://www.gamasutra.com/blogs/RHunterGough/20200117/355794/I_Hate_Roguelikes_And_So_Should_You.php (cit. a p. 5).

[12] Davide Aversa. *Procedural Contents Generation: History and techniques used in the modern video-game industry.* URL: https://www.davideaversa.it/wp-content/uploads/2015/06/Procedural-Contents-Generation.pdf (cit. a p. 6).

[13] Chris Baker. «'No Man's Sky': How Games Are Building Themselves». In: *Rolling Stones* (ago. 2016). URL: https://www.rollingstone.com/culture/culture-news/no-mans-sky-how-games-are-building-themselves-104779/ (cit. a p. 7).

[14] Ryan Kuo. «Why Borderlands 2 Has the Most Stylish Guns in Gaming». In: *The Wall Street Journal* (2012). URL: https://www.wsj.com/articles/BL-SEB-69805 (cit. a p. 7).

[15] Claudia Marchetto. *Epic Games acquisisce Quixel e rende gratuite più di 10.000 risorse di MegaScans per gli utenti Unreal Engine.* 2019. URL: https://www.eurogamer.it/articles/2019-11-12-news-videogiochi-epic-games-acquisisce-quixel-gratuite-10-000-risorse-megascans-utenti-unreal-engine (cit. a p. 9).

[16] SideFX. *Houdini help.* 2015. URL: https://www.sidefx.com/docs/houdini/index.html (cit. a p. 12).

[17] SideFX. *Spaceship - Houdini 12.1 to Houdini 15.5.* 2015. URL: https://www.orbolt.com/asset/SideFX::spaceship (cit. a p. 42).

[18] Holistic3d. *Generating a Procedural City with Unity.* 2016. URL: https://www.youtube.com/watch?v=xkuniXI3SEE (cit. a p. 44).

[19] Federico Pepe. *Random e Perlin Noise.* 2016. URL: https://blog.federicopepe.com/2016/06/random-vs-perlin-noise/ (cit. a p. 44).

[20] *Unity's interface.* URL: https://docs.unity3d.com/Manual/UsingTheEditor.html (cit. a p. 45).

[21] David R. Choffnes Harvey M. Deitel Paul J. Deitel. *Sistemi operativi.* Pearson, 2005, pp. 766, 838 (cit. a p. 47).

[22] *Houdini Engine for Unity - Session.* URL: https://www.sidefx.com/docs/unity/_session.html#Session_Architecture (cit. a p. 47).

[23] O. M. D'Antona James F. Kurose Keith W. Ross. *Reti di calcolatori e Internet. Un approccio top-down.* O. M. D'Antona, Pearson, 2008, pp. 146, 215 (cit. a p. 47).

[24] SideFX. *Houdini Engine for Unity - Houdini Assets*. URL: `https://www.sidefx.com/docs/unity/_assets.html` (cit. a p. 49).