POLITECNICO DI TORINO

Collegio di Ingegneria Informatica, del Cinema e Meccatronica

Tesi di Laurea Magistrale

Illuminazione e shading procedurale non fotorealistico in un prodotto di animazione indipendente

Relatore

Prof. Riccardo Antonio Silvio Antonino

Candidati

Giacomo Davide Massimo Balma

Andrea Lorusso

Dicembre 2019

Ad Anna, Eleonora e Rebecca.
Non stancatevi mai di lottare per quello che amate.
Comunque andrà, avrete un fedele alleato a guardarvi le spalle.

Andrea

Abstract

Nell'ambito del progetto di animazione Reverie Dawnfall verranno affrontati gli argomenti di shading procedurale e illuminazione di scene 3D.

In particolare, lo studio si concentrerà sulle tecniche di tipo non fotorealistico, con l'obiettivo principe di definire un paradigma estetico stilisticamente interessante e innovativo.

Nella prima parte saranno introdotti i concetti chiave necessari ad esporre gli esperimenti fatti: saranno confrontati i motori di rendering disponibili nel software Blender 2.80, considerandone fattori positivi e negativi.

Si passerà poi ad un'introduzione dei principi dello shading procedurale, esaminando in particolare i diversi sistemi di coordinate utilizzabili per la creazione di pattern e noise. Verranno quindi esposti i tentativi che hanno portato ai risultati finali, soffermandosi su confronti con lo stato dell'arte e analizzando ciò che può essere migliorato o approssimato con soluzioni real-time.

A seguire sarà trattato nel dettaglio lo sviluppo del plugin Light Studio per Blender. Si partirà dall'analisi delle Blender/Python API e dalla concezione stessa del plugin, procedendo con i dettagli tecnici dello sviluppo Python e concludendo su considerazioni riguardanti l'impatto che ha avuto (e che potrebbe avere) nel processo produttivo.

Successivamente verranno discusse tutte le scelte estetiche relative all'illuminazione degli ambienti costruiti, mostrando tentativi successivi e analizzando i parametri che hanno permesso di convergere verso il risultato finale.

Inoltre, durante il trattamento non saranno trascurate analisi di efficienza, sia computazionale che di produzione, considerando sempre il contesto in cui il progetto si sviluppa, ovvero un piccolo studio indipendente che muove i suoi primi passi nel campo dell'animazione 3D seriale.

In conclusione, verrà presentato lo stato di avanzamento dell'intero progetto, focalizzando l'attenzione su quanto messo a disposizione per gli eventuali progressi futuri in particolare nell'ambito del paradigma estetico e dello sviluppo sinergico degli strumenti necessari allo shading procedurale ed all'illuminazione 3D.

1 Introduzione

Il mondo dell'animazione da sempre regala a tutte le generazioni di spettatori una via di uscita dall'ordinario, un rifugio sicuro dove poter vivere senza timore in mondi fantastici avventure di valorosi eroi e fragili principesse (senza nulla togliere all'incommensurabile fascino dei fragili eroi e delle valorose principesse).

Negli ultimi anni, la forza espressiva del mezzo è stata consistentemente irrobustita dal ricco mondo della computer grafica che evolve a ritmi straordinari. La possibilità di sperimentare nuove tecniche e tecnologie a partire dalle solide fondamenta dell'animazione tradizionale ha regalato prodotti straordinari sotto ogni punto di vista.

Proprio nello spazio dove lo storytelling incontra i più raffinati mezzi tecnici viene concepito il presente progetto di tesi che si pone come obiettivo unico quello di curare la crescita di una fertile storia per poterne raccogliere i frutti.

La storia in questione è quella di Reverie Dawnfall, una serie animata le cui linee guida sono presentate nel seguente paragrafo.

1.1 Reverie Dawnfall

Reverie Dawnfall, ovvero "Il calare dell'alba su un sogno ad occhi aperti" è una serie d'animazione 3D cyberpunk che pone l'accento sulle tematiche dei disturbi neuro-psichiatrici, dell'esplorazione spaziale e del reale, dell'ecologia e le terribili conseguenze dell'ignorare lo sviluppo sostenibile. Protagonista della storia è un gruppo di ragazzi, disabili come la quasi totalità della popolazione mondiale, alle prese con un interrogativo che è al contempo letterale e filosofico: e se questo non fosse il mondo giusto? La serie è ambientata su di un pianeta dall'ecosistema al collasso e in rotazione sincrona rispetto alla propria stella. Un emisfero è quindi perennemente in ombra e uno illuminato. La città teatro della vicenda sorge sulla linea di demarcazione tra le due zone ed è pertanto in perenne illuminazione crepuscolare. Questo dà vita ad

una palette di colori che si distacca sia dal cupo notturno dei cyberpunk stile *Blade Runner*, sia dai saturi toni caldi dei post-apocalittici alla *Mad Max*.[1]

1.2 Situazione iniziale

Il progetto di Reverie Dawnfall nasce nel 2017 a cura di Robin Studio, uno studio creativo di Torino.

Dopo una prima copiosa fase di scrittura, concept e preproduzione si è incominciata una fase di ricerca e sviluppo per la definizione di un paradigma estetico interessante e dei mezzi necessari per perseguire il suddetto scopo.

In parallelo, per poter incominciare ad attrarre produttori e raccogliere fondi per produrre un episodio pilota, è stato realizzato lo storyboard per un teaser dalla durata di un minuto circa.

Dati i suddetti presupposti, Robin Studio ha deciso di formare un team composto da sei persone deputato al look development ed alla produzione dell'intero teaser.

Proprio in questo contesto è stato realizzato il presente progetto di tesi in azienda all'interno del quale verranno presentati, analizzati e discussi tutti i passi necessari per il raggiungimento della resa estetica definitiva in termini di shading e lighting e le esigenze produttive specifiche nei suddetti ambiti per la realizzazione di tutti gli shot del teaser.

Al momento dell'inizio del presente progetto erano stati realizzati:

- Il modello della protagonista privo di scheletro.
- I modelli di alcuni degli ambienti principali.
- Lo storyboard del teaser
- Alcuni test di shading

[1] http://robin.studio/reverie/ReveriePressbook.pdf

1.3 Obiettivi

Fin dall'inizio della lavorazione, la definizione degli obiettivi è stata estremamente chiara. Gli obiettivi preposti possono essere fondamentalmente riassunti in due macrocategorie:

- Obiettivi aziendali
- Obiettivi personali.

Per quanto riguarda gli obiettivi aziendali, è stato necessario, in prima battuta, presentare una serie di soluzioni estetiche che fossero coerenti con i capisaldi fissati in fase di concept in modo da scegliere la via da intraprendere. Una volta conclusa la fase di sperimentazione, ci si è dedicati alla realizzazione del teaser in modo da poterlo consegnare in concomitanza della conclusione del progetto di tesi.

Per quanto riguarda gli obiettivi personali, la maggiore sfida è stata quella di confrontarsi con la realtà di uno studio di produzione strutturato, naturalmente differente dall'organizzazione di progetti universitari precedentemente affrontati nel percorso accademico.

1.4 Outline

I capitoli saranno suddivisi come segue:

- **Stato dell'arte**: verranno esaminati brevemente i principali esempi di prodotti di animazione contemporanei che hanno esplorato l'ambito del non-fotorealistico.
- **Tecniche e strumenti**: nelle sezioni *EEVEE* e *Blender API* (Andrea Lorusso) ci si soffermerà sul confronto fra i motori di rendering path tracing e real-time, passando poi ad analizzare l'ecosistema di scripting interno al software Blender. Nella sezione *Shading procedurale* (Giacomo Balma) si introdurranno tutti i concetti necessari alla comprensione dei capitoli successivi.

- **Light Studio add-on** (Andrea Lorusso): nel capitolo sarà presentato *Light Studio,* il plugin per Blender realizzato per ottimizzare la configurazione delle luci in scena.
- **Shading** (Giacomo Balma): verranno esposti i diversi tentativi fatti nell'ambito dello shading, fino a giungere alla soluzione definitiva.
- **Lighting** (Andrea Lorusso): verranno presentate le configurazioni di luci utilizzate per tutti gli ambienti. Verranno commentate le scelte estetiche ed esplorati i processi che hanno permesso di convergere verso il risultato finale.
- **Risultati Finali**: saranno mostrati i risultati di rendering finali.
- **Conclusioni e prospettive future**: una breve riflessione su quanto realizzato, sui risultati e sulle prospettive future per il progetto.

2 Stato dell'arte

Durante la prima fase di analisi si è svolta una ricerca con lo scopo di raccogliere dei prodotti di riferimento che avessero un'estetica non-fotorealistica interessante. L'obiettivo era quello di vedere quali fossero le tecniche impiegate al momento, con il fine di acquisire nuove conoscenze e spunti per stabilire il look finale.

Il processo di ricerca di confronti con altri prodotti non si è esaurito durante la fase preliminare, al contrario si è prolungato per tutta la durata del progetto. Verranno riportati di seguito sia esempi di opere con data di uscita contemporanea allo sviluppo, sia esempi più vecchi.

2.1 Produzioni a basso budget: Gatta Cenerentola

Il primo prodotto preso in considerazione, da un lato per compatibilità estetica, dall'altro per il tipo di produzione, è stato il film d'animazione *Gatta Cenerentola*.

Gli aspetti interessanti sono diversi, primo fra tutti è il fatto che l'opera ricerca una veste grafica bidimensionale, nonostante sia interamente realizzato in 3D. Questo ha spinto ad approfondire le ricerche con lo scopo di capire quali fossero i principi alla base della riuscita dell'impresa.

Figura 1: un frame di Gatta Cenerentola

Fortunatamente, lo studio dietro alla creazione è molto attivo nell'ambito open source e basa la sua pipeline sul software Blender 3D.[2]

[2] https://www.blender.it/tag/gatta-cenerentola/

Continuando le ricerche, si sono trovate maggiori informazioni sulla produzione, individuando gli elementi caratteristici del prodotto.

- L'animazione è fatta a passo due.
- Vengono utilizzati degli overlay di acquerello animati sull'interezza del frame.
- Alcuni fondali sono dipinti digitalmente.
- I modelli 3D hanno un livello di dettaglio medio-basso, gran parte dei dettagli sono aggiunti con delle texture.
- Per camuffare l'aspetto discreto dei poligoni 3D viene applicato un displacement ai bordi degli oggetti in scena, con l'obiettivo di renderli morbidi.
- Viene aggiunta una rim light finta a tutta la scena basandosi su z-depth.
- L'animazione non è eccessivamente dettagliata, il basso frame rate unito a questa caratteristica danno una resa estetica simile a quella di un cartone animato.

Come si avrà modo di leggere in seguito, le informazioni raccolte si sono rilevate utili nel dare una direzione alla ricerca svolta, ma al contempo sono subito emersi dei punti di distacco che non si sarebbero potuti conciliare con le attuali esigenze della produzione. In particolare, l'assenza di un texture artist, l'assenza di persone in grado di creare fondali dipinti digitalmente e la scarsità di tempo disponibile per realizzare una grande quantità di lavoro manuale, hanno fatto sì che questi aspetti venissero messi da parte, concentrandosi di più sugli aspetti legati al compositing delle immagini.

2.2 Produzioni ad alto budget

In seguito all'analisi dei prodotti di basso budget, si è passati a considerare le opere più recenti e innovative, con la speranza di trovare qualche tecnica da introdurre in una produzione dal budget più basso adattandola e, se necessario, semplificandola per inserirla nella pipeline.

2.2.1 Spiderman – Into the Spiderverse

Il film d'animazione prodotto da Sony e distribuito a fine 2018 è senza ombra di dubbio un fuoriclasse dell'animazione degli ultimi anni. Il risultato è un compendio di scelte tecniche ed artistiche che hanno portato il film di Peter Ramsey, Robert Persichetti Jr. e Rodney Rothman ad emergere all'interno del sovrappopolato mondo cinematografico dell'immortale supereroe del Queens.

Figura 2: un frame di Spiderman into the spiderverse

Il film nasce con la chiara intenzione di voler perseguire un'estetica quanto più fedele possibile alle grafiche dei fumetti ed agli artbook realizzati in fase di preproduzione.

Dal punto di vista dell'animazione la ricerca è andata verso movimenti più stilizzati limitando gli effetti cartoon tradizionali (come ad esempio sqaush and stretch).

Altra soluzione particolarmente interessante, che ha permesso di aggiungere valore all'intero prodotto, è sicuramente stata la scelta di "dipingere" sui modelli delle ombre a seconda della geometria dell'oggetto stesso.

Per arricchire ulteriormente i modelli dei personaggi ed accentuare la loro espressività sono stati aggiunti degli accenti sul volto generati, in parte, da un

modello di machine learning addestrato sulla base dei lavori dei concept artists.

Infine è da sottolineare l'introduzione di linee e punti negli shader di quasi tutti gli oggetti in scena ed il color shifting per gli elementi fuori fuoco. Lo scopo perseguito è stato quello di accentuare ulteriormente l'effetto "carta stampata" (con i relativi difetti) con risultati davvero sorprendenti.

La maggior parte delle soluzioni finora descritte sono ovviamente state il frutto di mesi di ricerca e sviluppo e dunque difficilmente riproponibili in toto in una produzione indipendente. Tuttavia, prodotti come *Spiderman: into the spiderverse* sono certamente di ispirazione per la ricerca e la definizione di un canone estetico originale.

2.2.2 Love Death and Robots

L'ultima produzione che si è voluta analizzare è stata la serie animata Love Death and Robots. Il prodotto al momento conta 18 episodi, ognuno con uno stile diverso, uniti dal tema della fantascienza.

Figura 3: un frame di The Witness

La serie è paragonabile ad un vero e proprio catalogo di canoni estetici, raccogliendo un'infinità di spunti e tecniche. In particolare, fra tutti gli episodi spicca *The Witness*, ambientato in una metropoli, unisce elementi 2D e 3D senza apparenti transizioni, mantenendo una forte coerenza stilistica. Particolarmente notevoli sono gli overlay 2D realizzati in 3D, i quali aggiungono un livello di caratterizzazione unica alle mesh dei personaggi alle quali sono applicati.

Uno spunto di riflessione che può emergere riguarda la forte stilizzazione degli episodi e la necessità di variazioni. Alcuni stili particolarmente caratterizzanti possono infatti risultare inadatti ad una produzione seriale, in quanto non sopportabili da uno spettatore per più del tempo necessario allo sviluppo di un singolo episodio. Nel cercare di definire un canone estetico sarà quindi necessario considerare fattori di affaticamento e di fruibilità.

3 Tecniche e strumenti

L'analisi dello stato dell'arte, accompagnato dalle necessità di un progetto a basso budget, ha aiutato a restringere lo spettro delle opzioni disponibili per la produzione.

In particolare, visti i risultati notevoli ottenuti in *Gatta Cenerentola*, la scelta del software è ricaduta su Blender.
Quest'ultimo diventa quindi lo strumento centrale a partire del quale si snoda tutta la pipeline di produzione.

3.1 Blender e Blender API

Blender è una suite di creazione 3D gratis, open source e cross-platform: sono disponibili distribuzioni per Windows, Mac e Linux. Supporta la pipeline 3D nella sua interezza: modellazione, rigging, animazione, simulazione, rendering, compositing e motion tracking. Integra inoltre funzioni di montaggio video e animazione 2D.[3]

Il software vede la sua nascita nel 2000, anno in cui l'azienda *NaN*, fondata nel 1998, decide di realizzare un programma di creazione 3D interattivo. Tuttavia, a causa dei pessimi risultati di vendita, il progetto viene cancellato solamente due anni dopo.
Fortunatamente, anche grazie al supporto dato dalla crescente comunità di utilizzatori, Ton Roosendaal decide di dar vita alla fondazione no-profit Blender Foundation, acquistando così il software e rilasciandolo sotto licenza *GNU General Public License*. Da allora lo sviluppo non si è ancora arrestato: portato avanti da appassionati e professionisti, ogni giorno vengono aggiunte nuove funzionalità e strumenti.

[3] https://www.blender.org/about/

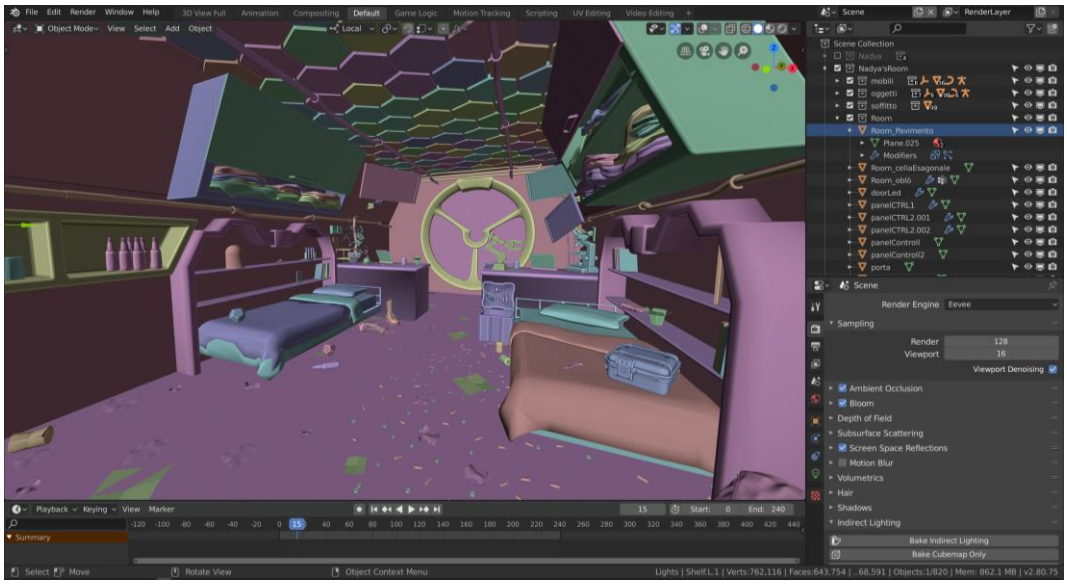


Figura 4: l'interfaccia di Blender 2.8

Vista la natura open source, durante la fase di prototipazione e sperimentazione ci si è trovati più volte a dover provare le nuove funzionalità implementate. In particolare, il progetto ha avuto la fortuna di vedere affrontati i quesiti legati alla resa estetica proprio durante la fase di rilascio della versione 2.80 di Blender. Come verrà ampiamente affrontato in seguito (EEVEE), questa introduce, oltre ad un completo rinnovo dell'interfaccia, un nuovo motore di rendering OpenGL real-time e fotorealistico: *Eevee*.
Nel corso delle prove saranno quindi analizzate sia le opzioni disponibili già da tempo che quelle nuove e in fase di Beta.

3.1.1 Blender API

Oltre tutti gli strumenti messi a disposizione per poter seguire l'intero processo produttivo all'interno di una singola applicazione, Blender mette a disposizioni dei propri utenti le Blender/Python API. Una simile risorsa consente, a chiunque si approcci al software, di gestire il flusso di produzione in modo molto agile.
Infatti, tramite le API è fondamentalmente possibile richiamare tutte le funzioni che l'interfaccia di Blender mette a disposizione e sviluppare tool che le

estendano, andando di fatto a creare nuovi strumenti per l'ottimizzazione della produzione.
Nel dettaglio, Blender, tramite la documentazione delle stesse API, chiarisce ciò che può essere ottenuto tramite lo scripting (attualmente basato su Python 3.x):

- Modifica di tutti i dati accessibili tramite interfaccia utente
- Modifica delle preferenze, delle key-map e dei temi
- Personalizzazione delle impostazioni per le esecuzioni dei tool
- Creazione di elementi dell'interfaccia grafica come menu, intestazioni e pannelli
- Creazione di nuovi tool
- Creazione di nuovi motori di rendering che si integrino con Blender
- Definizione di nuove proprietà per i dati presenti in Blender
- Creazione di geometrie tramite Python

I due modi più comuni per eseguire script Python in Blender sono tramite il Text editor interno a Blender o digitando i comandi direttamente nella console Python. Entrambi gli spazi sopra citati possono essere selezionati all'interno dello screen selector di Blender[4]. Per lo sviluppo di progetti più complessi è comunque possibile affidarsi ad ambienti di sviluppo propriamente strutturati (come ad esempio Visual Studio Code, Pycharm, ecc.).

Manipolazione dati e registrazione classi

Al fine di una maggiore comprensione dello sviluppo del plugin *Light Studio* (che sarà trattato nel *Light Studio* add-on), saranno introdotti alcuni concetti chiave per l'accesso e la manipolazione dei dati tramite le Blender/Python API.
Come primo passo, è possibile accedere ai dati presenti all'interno del file di Blender attualmente aperto attraverso il modulo `bpy.data`.

[4] https://docs.blender.org/api/

Una volta che si è avuto accesso al blocco di dati di interesse, come ad esempio l'elenco delle luci in scena (l'elenco dei materiali, delle collezioni ecc.), è possibile accedere ai loro attributi e modificarli (mostrato nel seguente esempio) esattamente come tramite i comandi dell'interfaccia grafica. Infatti, il *tooltip* di Blender, per ogni bottone, mostra l'attributo Python in questione per poter orientare l'utente nello scripting.

```
# Set "MyLight" energy to 2000
bpy.data.lights["MyLight"].energy = 2000.0
```

Il secondo passo di fondamentale importanza è il concetto di registrazione e de-registrazione di una classe.
Per ciascun pannello (o elemento dell'interfaccia grafica) che si desidera aggiungere via codice è necessario creare una nuova classe.
Tutte le nuove classi che vengono introdotte via codice devono necessariamente essere registrate (ed eventualmente de-registrate nel momento in cui non vengono più utilizzate).
Le funzioni `register()` ed `unregister()` sono le uniche funzioni che Blender chiama dal codice scritto dall'utente.
Le suddette funzioni generalmente compaiono in fondo allo script e vengono usate anche per altri scopi come ad esempio la preparazione dei dati per nuovi tool. Bisogna fare attenzione perché la funzione `register()` non verrà chiamata nuovamente quando sarà aperto un nuovo file .blend.
Le chiamate a `register()`/`unregister()` vengono anche utilizzate per attivare o disattivare add-on o ricaricare gli script mentre Blender è in esecuzione.
Nel caso in cui la chiamata a `register()` si trovi nel corpo dello script, la registrazione verrebbe effettivamente fatta al momento dell'importazione. In questo modo non ci sarebbe più distinzione tra l'importazione di un modulo ed il caricamento delle sue stesse classi in Blender.
Una volta registrata una classe, la sua definizione sarà caricata all'interno di

Blender e dunque diventerà disponibile insieme a tutte le funzionalità già esistenti nel software. Inoltre sarà possibile accedere alla stessa attraverso `bpy.types`, usando il *bl_idname* al posto del nome completo.
Di seguito è riportato un breve esempio di definizione di una classe e relativa registrazione.

```
import bpy

class NewPanel(bpy.types.Panel):
        bl_idname = "firstPanel"
        bl_label = "Hello World"
        bl_space_type = "PROPERTIES"
        bl_region_type = "WINDOW"
        bl_context = "object"

def draw(self, context):
        self.layout.label(text="Hello World")

bpy.utils.register_class(NewPanel)
```

3.2 EEVEE

Durante la fase di prototipazione e di definizione della resa estetica da attribuire alla serie Reverie Dawnfall ci si è più volte interrogati su quale potesse essere il motore di rendering che maggiormente avrebbe permesso di far convergere le intenzioni iniziali con l'effettiva resa del prodotto finito.
Fino a qualche mese fa, Blender offriva nativamente *Cycles*, motore di rendering physically-based (PBR) che offre un notevole controllo dal punto di vista artistico, grazie ad un relativamente semplice sistema di shading gestito da un editor a nodi che permette di conferire grande flessibilità alla produzione. Tuttavia, trattandosi di un sistema basato su raytracing, comporta un notevole onere computazionale.

A luglio 2019, dopo un lungo periodo di sperimentazione Blender ha rilasciato la versione 2.8 del software, introducendo un nuovo motore di rendering: EEVEE.

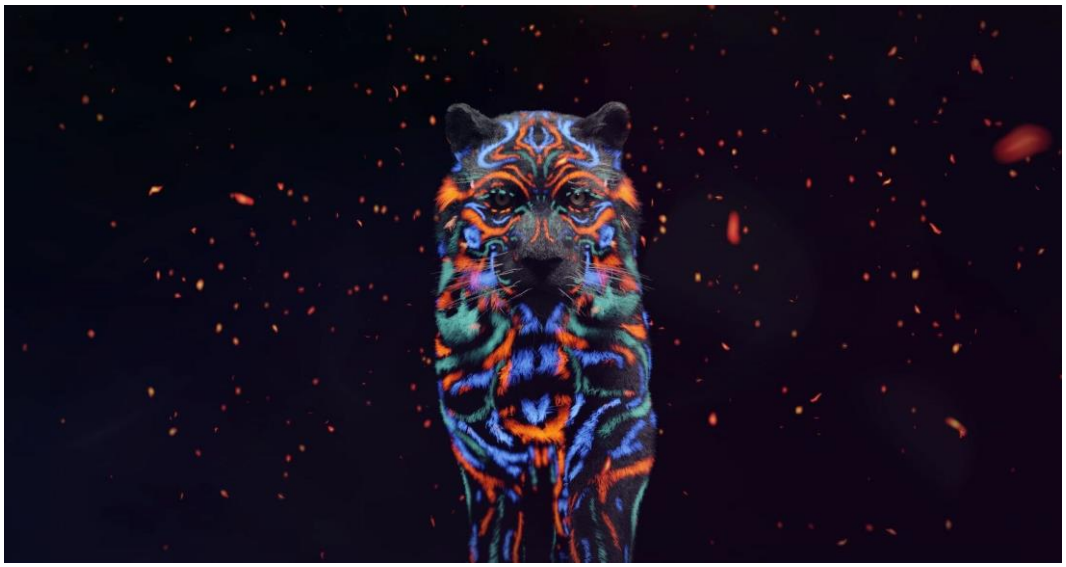


Figura 5: modello 3D con shading, lighting e rendering realizzati in EEVEE

EEVEE è un motore di rendering realtime basato su OpenGL che fa della rapidità nella risposta e dell'interattività la sua principale cifra tecnica. Oltre a rivelarsi uno strumento estremamente valido per la resa di contesti fotorealistici (seppur non all'altezza di motori di rendering basati su raytracing),

sono stati introdotti nel sistema a nodi già precedentemente citato delle nuove risorse.
Una delle nuove funzioni introdotte con EEVEE, che ha rappresentato un discriminante per le scelte di produzione, è stata l'introduzione del nodo *Shader to RGB*. Il suddetto nodo consente, a differenza di quanto disponibile in Cycles, di rompere la catena fotorealistica grazie alla possibilità di "intercettare" i valori RGB in uscita da un nodo di tipo *shader* (Glossy, Diffuse, Principled BSDF, ecc.) e poterli elaborare per poi far nuovamente convergere il tutto verso un nodo shader (non necessariamente uguale a quelli precedenti).

Date le promettenti premesse, si è deciso quindi di incominciare a condurre alcuni esperimenti su EEVEE quando ancora la versione 2.8 di Blender (e dunque anche il motore di rendering) era ancora in fase beta.
La buona riuscita dei suddetti test, unita a tutto quanto detto finora, ha contribuito a far sì che la scelta del motore di rendering definitivo per il tipo di estetica che si desidera per Reverie fosse proprio EEVEE.

3.2.1 Real time vs path tracing

Come già precedentemente accennato, la differenza tra Cycles ed EEVEE in fase di rendering è una differenza strutturale.
I motori di rendering basati su path tracing (come ad esempio Cycles) aspirano a riprodurre in modo quanto più fedele possibile il mondo reale ed i fenomeni fisici di riflessione, rifrazione e dispersione della luce attraverso il tracciamento dei "raggi" che ipoteticamente rappresentano la traiettoria dei fotoni.
Di fatto, si decide arbitrariamente un numero di "raggi luminosi" che partono dalla camera e vengono tracciati fino a quando non intercettano una sorgente luminosa o il "cielo". Ogni volta che un raggio "colpisce" un oggetto si presentano una molteplicità di scenari: a seconda del materiale assegnato all'oggetto che viene intercettato, il raggio può essere parzialmente trasmesso o parzialmente riflesso con intensità inferiore rispetto all'istante precedente

all'impatto (come mostrato nella *Figura 6*).
Andando a seguire la traiettoria di tutti i raggi luminosi, si ha la possibilità di descrivere in maniera piuttosto accurata la distribuzione dell'energia luminosa all'interno della scena 3D e dunque procedere con la rasterizzazione.
È possibile definire l'accuratezza del metodo di path tracing andando a specificare il numero di raggi da far propagare all'interno dello spazio 3D e definendo il numero massimo di rimbalzi del raggio prima di poter considerare trascurabile la componente di energia luminosa.

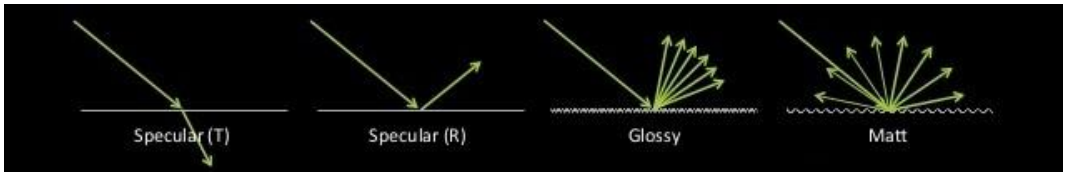


Figura 6: interazione tra luce e oggetti di diversi materiali.

Differentemente da quanto detto finora, EEVEE è un *rasterizer*, un motore di rendering real-time basato su OpenGL e compatibile con la creazione di materiali che supportano il *Physically Based Render* (PBR).
I rasterizer in prima battuta raccolgono informazioni sulla geometria degli oggetti che ricadono all'interno del *frustum visivo*, poi ad ogni vertice dei poligoni viene assegnato un colore. Successivamente viene fatta una conversione dei triangoli (o dei poligoni che compongono le geometrie degli oggetti) in pixel sullo schermo 2D. A ciascun pixel viene assegnato un colore iniziale corrispondente al valore memorizzato nel vertice che ricade all'interno del determinato pixel.
Successivamente, i valori iniziali di colore vengono ulteriormente processati (durante la fase di *shading*) e vengono aggiornati a seconda di come l'illuminazione della scena "colpisce" il contenuto di ciascun pixel ed in base alla\e texture applicate ai vari oggetti in scena.
Oltre a quanto detto, EEVEE supporta anche sistemi per il bake della componente di luce indiretta, per le screen space reflections, l'ambient occlusion ed

altre funzioni per provare a colmare la distanza con la resa dei motori di rendering basati su path tracing, pur mantenendo la velocità e la reattività di un motore real-time.

Nei paragrafi successivi ci si pone come obiettivo quello di analizzare punto per punto alcune delle principali differenze tra i due metodi alternativi di renderizzare una scena 3D.

Illuminazione globale

La componente di illuminazione globale (o illuminazione indiretta) è intrinseca nel concetto di raytracing in quanto i "raggi luminosi" acquisiscono dati (componente di colore assorbita, quantità di energia assorbita e trasmessa, ecc.) ogni volta che viene registrato il "rimbalzo" su un oggetto. Le suddette informazioni vengono utilizzate per definire il comportamento dell' i-esimo raggio all'urto contro la successiva superficie.
Con EEVEE, ovviamente, questo tipo di informazioni non possono essere calcolate, e dunque sono stati introdotti degli strumenti chiamati *Light probes* che permettono di approssimarne il valore.
I light probes "scattano una fotografia" della scena dal loro punto di vista nel mondo e proiettano i colori corretti sugli oggetti nell'area di loro influenza.
La componente della luce indiretta derivante dai light probes deve essere calcolata offline e dunque non funziona in caso di sorgenti luminose animate.
Chiaramente ad un crescente numero di light probes corrisponde una migliore approssimazione del comportamento della luce ma anche un crescente onere computazionale in fase di bake.

Figura 8: test illuminazione globale con Cycles

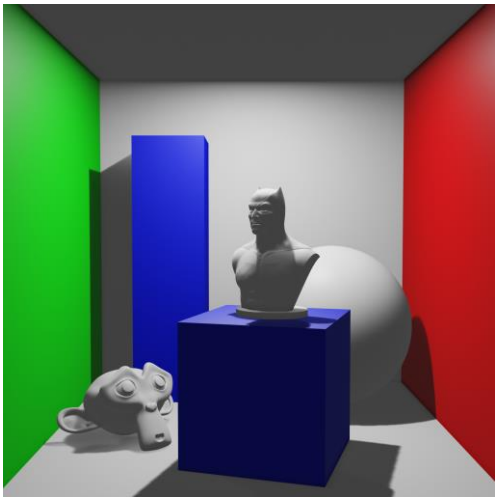


Figura 8: test illuminazione globale con EEVEE

Nei test mostrati si può notare come emergano due delle principali differenze tra i motori di rendering. In EEVEE le ombre sono più nette e tendenzialmente omogenee, al contrario, in Cycles, le ombre sono più morbide e "spalmate" gradualmente lungo le superfici. Inoltre, in Cycles è netto il contributo della luce indiretta che di fatto influisce sul colore degli altri oggetti in scena. Di seguito (*Figura 9*), è riportato anche il medesimo esempio della scena precedente renderizzato con EEVEE dopo l'inserimento di un light probe ed il conseguente bake della luce indiretta.

Figura 9: test illuminazione globale con EEVEE dopo il bake della luce indiretta

Riflessioni

Le riflessioni sono uno dei punti dove maggiormente emerge la differenza tra i due approcci al rendering.

Come già detto, Cycles calcola tutto a partire della propagazione dei raggi luminosi, e quindi la straordinaria resa delle riflessioni è esattamente quello che ci si aspetta.

Per realizzare riflessioni sufficientemente credibili, EEVEE (come tutti gli altri rasterizer) implementa alcuni stratagemmi per imitare i risultati ottenuti con il path tracing, con discreti risultati.

Il metodo utilizzato è quello delle *Screen space reflections*: l'immagine risultate di un oggetto viene presa e ruotata a seconda della direzione della riflessione. Questa soluzione risulta estremamente rapida e fornisce una resa credibile nella maggior parte delle circostanze. Il suo difetto principale è che non funziona con tutti gli oggetti (o parte degli oggetti) che non sono in camera.

Come nel caso dell'illuminazione globale, anche per le riflessioni, sono stati introdotti i *Reflection probes*, che non sono altro che degli oggetti (piani, sfere, cubi ecc.) che fungono da camere virtuali. Essi sono in grado di renderizzare ciò che li circonda e successivamente mappano l'immagine ottenuta sull'oggetto stesso. Reflection probes planari sono semplici da calcolare e quindi possono essere renderizzati in real-time, al contrario, mappe cubiche

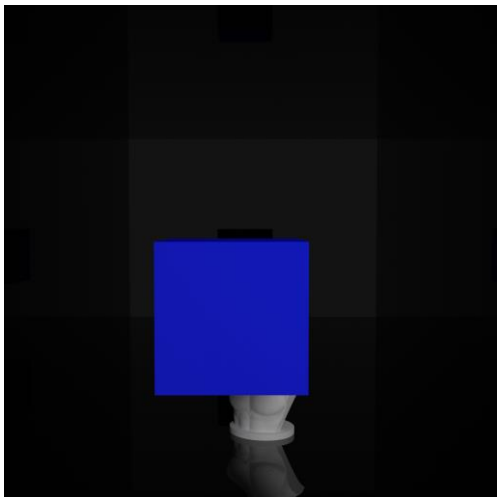


Figura 11: test riflessioni in Cycles

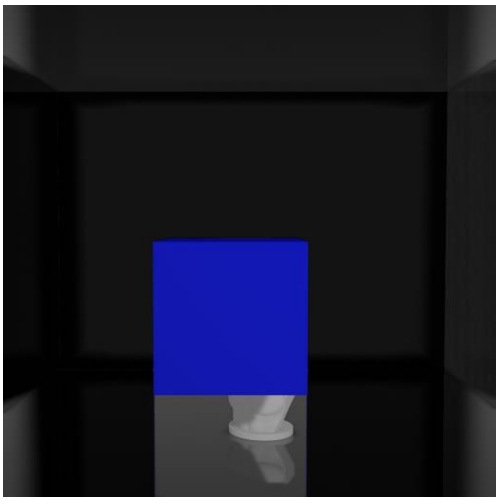


Figura 11: test riflessioni in EEVEE

e sferiche necessitano di un maggiore tempo di calcolo (e dunque di una fase di baking). Inoltre, trattandosi di un calcolo offline, gli oggetti dei quali vengono acquisiti i dati non possono essere animati.

Le immagini di test precedenti (*Figura 11* e *Figura 11*) mostrano come gli elementi del busto occlusi dal cubo siano di fatto esclusi dal calcolo delle riflessioni (essendo esclusi dallo screen space). Inoltre, si può notare come EEVEE introduca degli artefatti piuttosto evidenti nel riflesso del cubo blu (come mostrato anche nelle immagini di test successive (*Figura 13* e *Figura 13*).

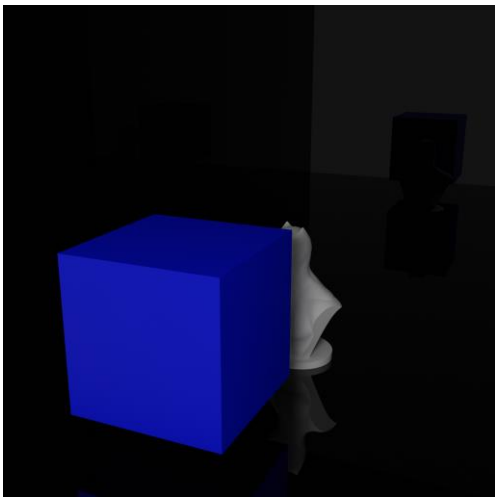


Figura 13: test_2 riflessioni in Cycles

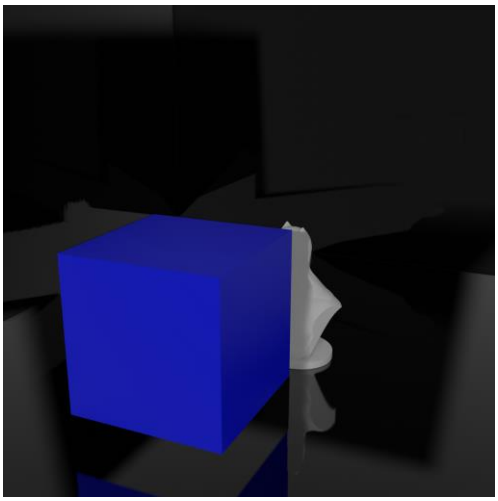


Figura 13: test_2 riflessioni in EEVEE

Per ottenere un quadro completo, è stato condotto un test inserendo quattro reflection plane sulle pareti della stanza ed è stato fatto il bake delle riflessioni in EEVEE (*Figura 14*).

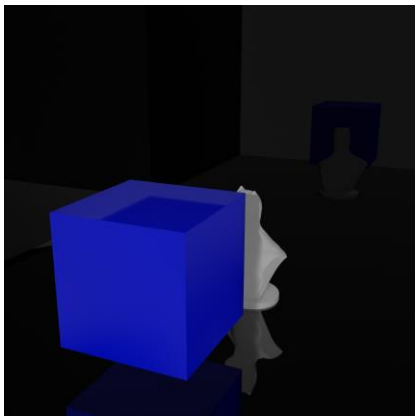


Figura 14: test_2 riflessioni in EEVEE con reflection planes

Rifrazioni

Le rifrazioni in EEVEE vengono implementate distorcendo ciò che si trova dietro gli oggetti (o visti da una cubemap). La quantità ed il tipo di distorsione che viene impressa dipendono dalla disposizione delle normali e dallo spessore degli oggetti occlusi. Questo significa che la luce non rimbalza all'interno dell'oggetto "trasmissivo" come in Cycles, e soprattutto non si ha alcun tipo di informazione riguardo gli angoli di incidenza dei raggi luminosi con gli oggetti al momento dell'impatto.
I risultati sono quasi completamente paragonabili nel caso di oggetti con geometrie semplici che rifrangono la luce. Per oggetti dalla geometria più complessa incominciano ad emergere le differenze tra i rasterizer ed i motori di rendering basati su raytracing.

Figura 16: test rifrazioni in Cycles

Figura 16: test rifrazioni in EEVEE

Le immagini di test (*Figura 16* e *Figura 16*) mostrano come la raffinatezza dei dettagli di Cycles sia nettamente superiore al risultato ottenuto con EEVEE. Quest'ultimo, infatti, introduce un gran numero di artefatti, oltre a non trasmettere oltre il modello dell'oggetto trasparente alcuna componente di energia luminosa (creando dunque un'ombra molto netta sulla parete posteriore).

Trasparenza

Entrambi i motori di rendering supportano i materiali trasparenti. La trasparenza in Cycles è limitata dal numero di rimbalzi dopo i quali il materiale verrà renderizzato come nero puro (altresì quando l'energia luminosa residua può essere considerata trascurabile).
In EEVEE l'impostazione di *Alpha Blend* consente di rappresentare la trasparenza "stratificando" i materiali uno sull'altro, dal più vicino alla camera, al più lontano. È necessario impostare un limite oltre il quale si ferma la stratificazione e gli oggetti smettono di essere renderizzati altrimenti si rischia di ottenere un risultato confusionario e poco definito.
In generale, la resa dell'alpha clip implementato per EEVEE fornisce degli ottimi risultati. Il compromesso che però bisogna accettare è che i materiali possono essere o completamente opachi o completamente trasparenti, non è supportata un'unione dei due. Inoltre, i materiali tendono a risultare più trasparenti man mano che ci si allontana dalla camera.

Ambient occlusion

L'ambient occlusion è un metodo di ombreggiatura che contribuisce a conferire realismo ai modelli di riflessione locale in quanto tiene conto dell'attenuazione luminosa in prossimità dei volumi occlusi.[5]
In Cycles l'ambient occlusion si basa sulla distanza tra le superfici nello spazio 3D. Al contrario, in EEVEE, viene considerata la distanza tra le facce degli oggetti nello spazio 2D dell'immagine che viene mostrata. Il risultato è comunque discreto poiché nel calcolo viene considerata la *depth pass* (pass di render che tiene conto della profondità degli oggetti in scena a partire dal piano della camera/osservatore) per determinare dove far scomparire la componente di ambient occlusion evitando di calcolarla per oggetti troppo distanti.

[5] https://it.wikipedia.org/wiki/Ambient_occlusion

Figura 18: test ambient occlusion Cycles

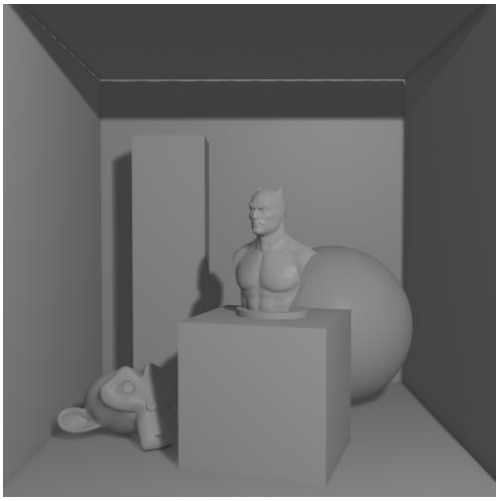


Figura 18: test ambient occlusion EEVEE

Ombre

Le ombre in EEVEE vengono realizzate catturando un'immagine dal punto di vista di ciascuna sorgente di luce, e verificando se un determinato pixel fa parte o meno di ciascuna delle immagini precedentemente raccolte (tecnica di *Shadow mapping*). Maggiore sarà la "risoluzione" dell'ombra, maggiore sarà il numero di pixel da verificare e maggiore sarà l'accuratezza della transizione da zone di luce a zone di ombra.

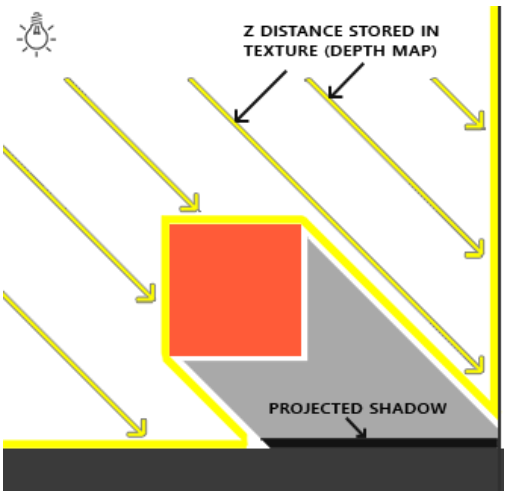


Figura 19: Schematizzazione del funzionamento dello Shadow mapping

Le impostazioni di Soft shadows e contact shadows aiutano a colmare le lacune delle tecniche di shadow mapping, ma rinunciando alla naturalezza dei rimbalzi della luce di Cycles, risulta abbastanza complesso ottenere immagini con una resa più fotorealistica.

Volumi

Una delle poche, se non l'unica, caratteristica in cui EEVEE si impone decisamente su Cycles è la rappresentazione degli effetti volumetrici e le luci volumetriche. Le luci volumetriche sono una particolare tecnica della computer grafica attraverso la quale è possibile vedere raggi o fasci di luce molto netti attraversare l'ambiente. È, ad esempio, il caso di luce diurna molto forte che penetra da una relativamente piccola apertura o magari l'effetto di un lampione in un esterno nebbioso.
Per poter funzionare nei motori di rendering basati su raytracing, l'illuminazione volumetrica necessita di due informazioni: la shadow map delle luci presenti in scena e le informazioni di profondità (depth map). A partire dal *near clipping plane*, l'intera scena viene scannerizzata e vengono presi dei valori campione memorizzati in un buffer dedicato. Per ogni campione, si determina se esso è illuminato direttamente o meno usando le shadow map. Solo i campioni direttamente illuminati contribuiranno al valore finale del pixel.[6]
In EEVEE sono ovviamente necessarie delle semplificazioni: nella fattispecie la dispersione e la diffrazione tipica delle luci volumetriche viene simulata valutando i volumi degli oggetti presenti all'interno del frustum. Per realizzare l'effetto finale vengono usate una serie di texture 3D che hanno un grosso impatto in termini di impegno della memoria. è possibile manipolare il reale impatto delle texture manipolando i parametri di *Tile size* e *Samples*. Nonostante la superiorità di EEVEE nella resa, bisogna essere consapevoli di accettare una serie di compromessi:

[6] https://en.wikipedia.org/wiki/Volumetric_lighting

- I volumi sono renderizzati solo a partire dai "raggi" provenienti dalla camera. Non forniscono alcun contributo al fine del calcolo dei e *Refraction probes*.
- I volumi non ricevono un contributo in termini di illuminazione dagli *Irradiance volumes*, ma lo ricevono dall'illuminazione diffusa del "mondo".
- L'illuminazione volumetrica non rispetta le reali forme delle luci. Tutte le sorgenti in scena verranno considerate come *Point light*.

Figura 21: test luci volumetriche Cycles

Figura 21: test luci volumetriche EEVEE

Subsurface scattering

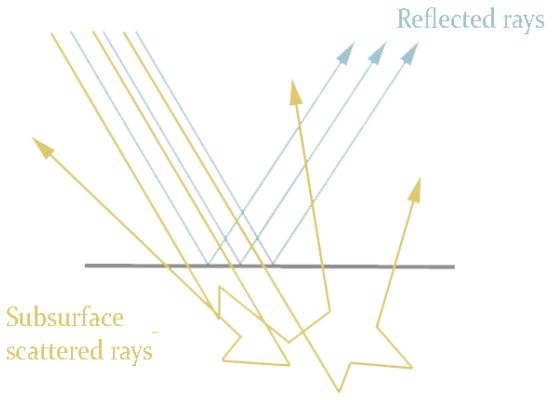


Figura 22: Schematizzazione Subsurface scattering

Il subsurface scattering è un meccanismo di trasporto della luce per il quale la luce penetra la superficie di un oggetto semi-trasparente e viene diffratta in base alle caratteristiche del materiale stesso ed esce dalla superficie in un punto diverso rispetto al contatto iniziale. Generalmente la luce penetra e subisce una serie

di riflessioni successive prima di uscire nuovamente dalla superficie. Il subsurface scattering rientra, insieme ad Ambient occlusion e riflessioni, nella categoria di informazioni calcolate in screen space ed in real-time per cercare di approssimare fenomeni naturalmente integrati nei motori di rendering basati su raytracing.

Figura 24: test subsurface scattering Cycles

Figura 24: test subsurface scattering EEVEE

Profondità di campo

EEVEE supporta la profondità di campo, e permette di ottenere risultati molto interessanti. Anche in questo caso si tratta di un'approssimazione, al contrario di Cycles che è rigorosamente fedele alle regole dell'ottica geometrica. L'unica reale differenza che si può apprezzare risiede nelle zone di transizione tra le aree nitide e le aree dove comincia la sfocatura ed in casi di distorsioni anamorfiche.

Nella maggior parte dei casi, EEVEE fornirà ottimi risultati in tempi anche nettamente inferiori. Inoltre, Blender, ha promesso significative migliorie della gestione della profondità di campo in EEVEE nelle prossime versioni del software che dovrebbero portare il motore real-time a colmare quasi completamente il divario con Cycles.

Figura 26: test profondità di campo Cycles

Figura 26: test di profondità di campo EEVEE

Motion blur

Cycles supporta il motion blur della maggior parte degli oggetti e degli elementi della scena, mentre EEVEE, al momento della stesura, supporta esclusivamente il motion blur della camera. Gli oggetti in movimento o le armature che si deformano non avranno alcun effetto.
Tuttavia, le suddette limitazioni non sono strutturali per i rasterizer e dunque potrebbero essere integrate in EEVEE in futuro. Agli esordi, anche in Cycles non erano implementati e sono stati significativamente migliorati con il passare delle versioni.

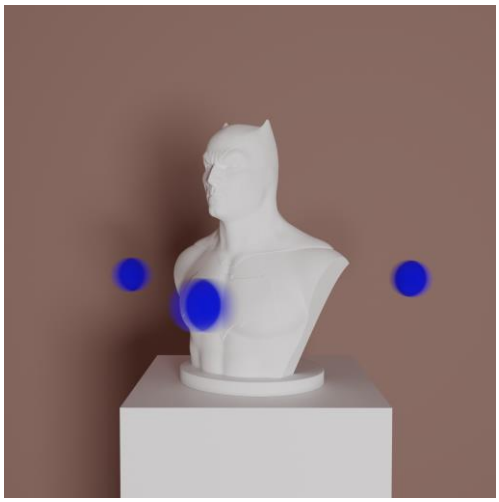


Figura 28: test motion blur Cycles

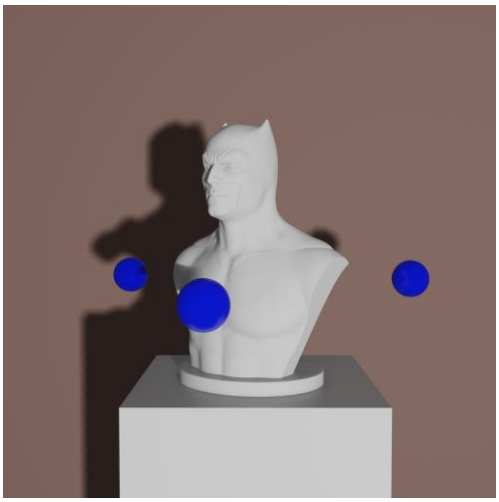


Figura 27: test motion blur EEVEE

I primi test (*Figura 28* e *Figura 27*) sono stati condotti con la camera ferma e le sfere in movimento. Per questo motivo nel test con EEVEE non si nota alcun tipo di sfocatura. Nelle seguenti immagini (*Figura 27* e *Figura 29*) sono invece riportati altri due test in cui sia la camera che le sfere sono in movimento in modo da poter apprezzare anche la leggera sfocatura sul busto.

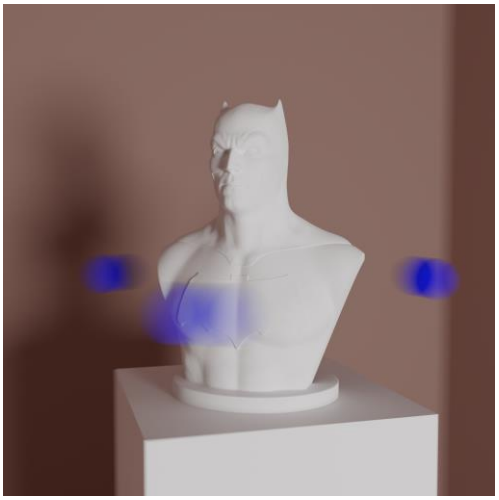


Figura 29: test_2 motion blur Cycles

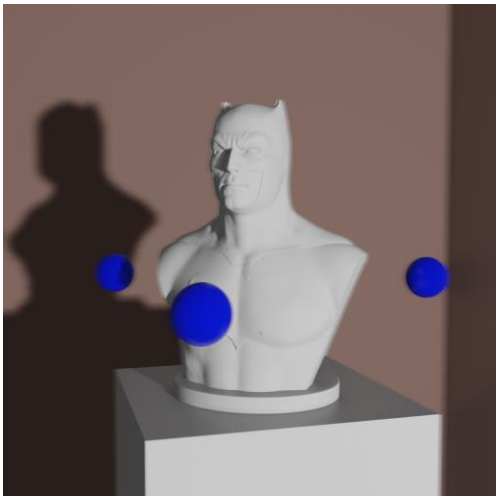


Figura 30: test_2 motion blur EEVEE

Passi di render

L'ultima differenza significativa da riportare riguarda la possibilità di esportare le diverse pass di rendering, particolarmente utili in caso di operazioni di compositing. Al momento della scrittura, EEVEE permette di esportare le pass più importanti: Mist, Normal ed Ambient Occlusion. Mancano ancora alcune molto importanti come Vector, Object, Material Index, Shadow, Emission e Cryptomatte. Potrebbero essere anche essere tutte integrate col passare del tempo ma comunque non permetterebbero di ottenere un controllo fine in fase di post-produzione a causa della mancanza delle informazioni relative ai raggi luminosi.

3.3 Shading procedurale

Volendo mantenere un flusso di lavoro che permettesse di gestire con flessibilità le diverse prove che si sarebbero dovute fare per trovare l'estetica finale, viste anche le incertezze su quest'ultima, si è deciso di procedere con un metodo che consentisse di massimizzare le possibilità creative, costruendo al contempo un ambiente in grado di essere utilizzato in futuro dagli artisti che avrebbero dovuto occuparsi delle fasi di shading, senza che quest'ultimi necessitassero competenze tecnico-artistiche specifiche.

A sommarsi alle necessità di cui sopra, se ne aggiungono alcune dovute all'attuale formazione del team di lavoro:

- Assenza di un *texture artist*, ovvero qualcuno che si occupi della realizzazione di texture da applicare ai modelli 3D.
- Produzione di tipo seriale, quindi con un grande numero di inquadrature ed episodi, nonostante la dimensione ridotta del gruppo di persone al lavoro sul progetto.
- Necessità, a regime di produzione, di poter realizzare episodi il più rapidamente possibile e con il minimo intervento possibile in post-produzione.

Cercando di soddisfare quanto sopra illustrato, si è quindi deciso di affrontare il problema utilizzando un metodo di shading procedurale.

3.3.1 Metodo procedurale

Prima di analizzare l'approccio procedurale nell'ambito dello shading è interessante soffermarsi introducendo i concetti e le modalità caratterizzanti generali, considerando le principali differenze con le tecniche standard.

L'origine del termine

Il termine *procedura* è comunemente noto come

> Il modo di procedere, cioè di operare o di comportarsi in determinate circostanze o per ottenere un certo risultato.[7]

In informatica questo assume una particolare accezione, individuando un particolare paradigma di programmazione. La *programmazione procedurale* consiste nel creare dei blocchi di codice sorgente, identificati da un nome e racchiusi da dei delimitatori, che variano a seconda del linguaggio di programmazione; questi sono detti anche sottoprogrammi, procedure o funzioni, a seconda del linguaggio e dei loro ruoli all'interno del linguaggio stesso.

Si consideri per esempio il seguente blocco di pseudocodice:

```
Type nomeFunzione(input) {
    // Dichiarazioni e Istruzioni
    return output;
 }
```

Quest'ultimo è caratterizzato da dei parametri di ingresso (*input*), da un gruppo di istruzioni da eseguire, e da un'uscita (*output*), la quale solitamente dipende da input.

Ciò che è caratterizzante dunque è la delegazione del processo risolutivo ad un algoritmo. I vantaggi durante le situazioni di ripetitività sono ovviamente notevoli.

[7] http://www.treccani.it/vocabolario/procedura/

Affrontare il problema

Introdotto il concetto, si possono ora esaminare le modalità. Dato un problema, per esempio la realizzazione di un modello 3D complesso, si possono individuare due fasi principali per il suo superamento:

- analisi;
- sintesi.

Si noterà come il paradigma procedurale ha molto in comune con il metodo scientifico-sperimentale, costruendo un profondo collegamento fra matematica e arte visiva.

Fase di analisi

La prima parte è quella più importante e corposa. È infatti quella dalla quale dipende l'efficacia effettiva della soluzione.

Durante questa fase vanno identificate le caratteristiche del problema, in particolare quelle che potremmo definire come le sue simmetrie.

Per simmetria si intende un attributo che si manifesta ripetutamente durante l'analisi. La ripetitività è proprio ciò che permette di semplificare la soluzione, ed è quindi il fulcro della ricerca.

Individuate le simmetrie del problema si può procedere ad una modellizzazione di alto livello del fenomeno. Attraverso un processo di astrazione e, in alcuni casi, di approssimazione, si giunge alla definizione dei parametri di input e alle istruzioni necessarie alla generazione dell'output.

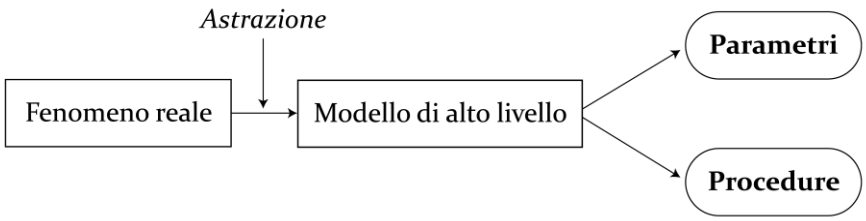


Figura 31: processo di analisi

Ritornando alla premessa, e tenendo presente che l'obiettivo è quello di ottenere una soluzione veloce e flessibile, è importante in questo momento considerare quelle che potrebbero essere le condizioni al contorno. Si vuole massimizzare la flessibilità offerta dallo strumento che si andrà a realizzare. Al contempo non lo si vuole però appesantire in modo eccessivo con parametri superflui.

Tradotto nella prospettiva di un piccolo studio creativo, è questo il momento in cui vanno considerate le eventuali possibili modifiche che il cliente potrebbe richiedere. Definendo i parametri di input (e le conseguenti operazioni) in modo opportuno avremo la possibilità, con il solo inserimento di que-sti, di ottenere i cambiamenti desiderati sul prodotto finale.

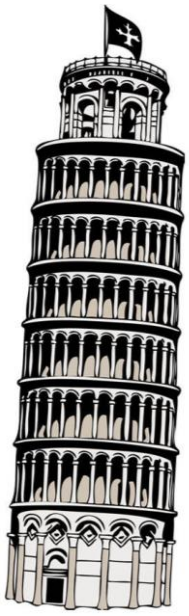
Si passa dunque alla fase di realizzazione del codice in grado di compiere le operazioni necessarie. Al termine di questa fase si otterrà uno strumento in grado di ricavare in modo automatico quanto richiesto.

Fase di sintesi

Fino a questo momento non si è ancora ottenuto un risultato tangibile, soprattutto a livello di quantità di materiale generato. È infatti durante la fase di sintesi che avverrà il concepimento del prodotto finale. Inserendo i parametri in ingresso opportunamente scelti, si otterrà l'output desiderato.

Figura 32: processo di sintesi

Esempio pratico: la Torre di Pisa



Si presenta di seguito un breve esempio pratico per chiarire il procedimento finora descritto.

Si ipotizzi di voler realizzare un modello 3D della Torre di Pisa. Si trascuri la sua pendenza caratteristica e ci si concentri sulla ricerca delle sue simmetrie. Focalizzandosi in particolare sul secondo registro salta subito all'occhio la sua forte caratterizzazione a moduli.

Si prosegue dunque all'identificazione dei diversi livelli di simmetria locale, partendo da una veduta macroscopica e muovendoci verso una più specifica:

- nel registro centrale vi sono sei piani pressoché identici;
- ogni piano è costituito da una successione di arcate disposte lungo una circonferenza di base;
- ogni arcata è formata da una colonna e da un arco;
- [...]

Il procedimento analitico può essere iterato fino al dettaglio più specifico, si potrebbe considerare il capitello della colonna, il numero di scanalature del fusto, ecc.

È importante stimare il livello di flessibilità che il nostro modello dovrà avere. L'obiettivo è rendere statici gli elementi che, viste le specifiche di progetto, si pensa non necessiteranno delle modifiche. Per semplicità ci si limiterà a considerare i primi tre livelli identificati.

L'arcata

Si inizia quindi dalla simmetria più specifica muovendosi verso quella globale.

Si procede a modellare il modulo di base, come riportato nella *Figura 33*: l'arcata a destra. Di questa si definiscono le variabili di input: `altezzaColonna`, `larghezzaArcata`. Agendo su queste sarà possibile modificare le proprietà del modulo, in questo modo si potranno ottenere potenzialmente infinite varianti dello stesso.

Figura 33: l'arcata

Lo pseudocodice associato sarà quindi:

```
Arcata generaArcata(altezzaColonna, larghezzaArcata) {

    // Genera il modello date le specifiche di ingresso
    return arcata;
}
```

Figura 34: alcune varianti di arcata ottenibili

Il piano

Il singolo piano della Torre è costituito da una successione (vettore) di arcate disposte lungo una circonferenza caratterizzata dalla variabile raggio.

L'algoritmo inoltre riceverà in ingresso il numero di arcate del piano con la variabile `numeroArcate`.

Figura 35: il piano

```
Piano generaPiano(raggio, numeroArcate, altezzaColonna) {

    // Stabilisce il valore di larghezzaArcata in modo che
    // le arcate abbiano la larghezza corretta per essere
    // disposte lungo la circonferenza di raggio.

    arcata = generaArcata(altezzaColonna, larghezzaArcata);

    // Genera il piano da una composizione di moduli arcata
    return piano;
}
```

Figura 36: alcune varianti di piano

Il registro

Un registro è composto dalla sovrapposizione dei diversi piani. L'unica variabile caratteristica di nostro interesse sarà dunque `numeroPiani`.

```
Registro generaRegistro(numeroPiani, raggio, numeroArcate,
altezzaColonna) {

    piano = generaPiano(raggio, altezzaColonna, larghezzaAr-
cata);

    // Genera il registro da una sovrapposizione di moduli
    // di tipo piano sull'asse verticale.
    return piano;
  }
```

Figura 37: il registro

Si noti come ognuna delle funzioni descritte, per comodità di rappresentazione, in pseudocodice richiami quella precedentemente definita. Questo proprio a causa del lavoro di analisi con il quale si è individuata una cellula modulare elementare sulla quale operare.

Definendo opportunamente le variabili `numeroPiani`, `raggio`, `numeroArcate`, `altezzaColonna` sarà possibile generare in modo procedurale una Torre di Pisa.

Il vantaggio è che se dovessimo apportare delle modifiche al numero di colonne, o alla loro dimensione, o al numero di piani, questo sarà possibile senza dover affrontare nuovamente tutta la modellazione da zero.

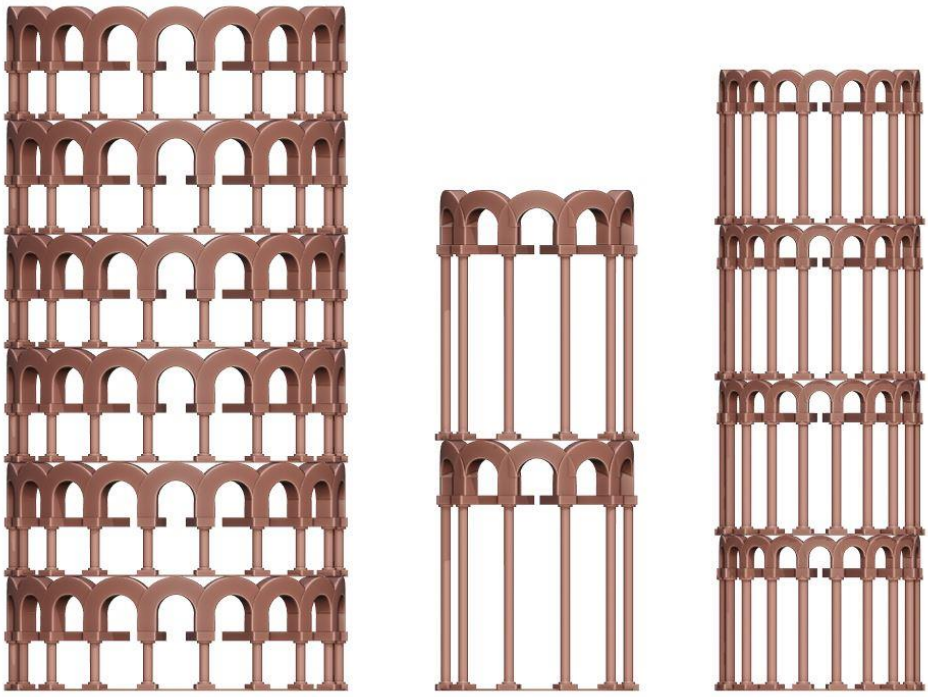


Figura 38: alcune varianti ottenibili con l'algoritmo descritto

Metodo procedurale applicato allo shading

Una volta chiarito il meccanismo di base, non resta che applicarlo allo shading. Ci si starà chiedendo, giunti a questo punto, come quanto sopra descritto possa venire in aiuto nell'affrontare le problematiche legate a quest'ultimo.

La risposta è molto semplice e trova le radici nella necessità di rappresentare pattern e texture complessi senza interventi manuali. La creazione questi può essere risolta tramite implementazione di algoritmi che, a partire da coordinate 3D e dei parametri di input predefiniti, riescono a restituire automaticamente risultati che necessiterebbero intere giornate di lavoro per essere realizzati da un texture artist sui singoli modelli o da un animatore 2D sui frame renderizzati.

3.3.2 Ambiente di shading interno a Blender

Blender mette a disposizione un intero ambiente dedicato alla creazione dei materiali: lo *Shader Editor*[8].

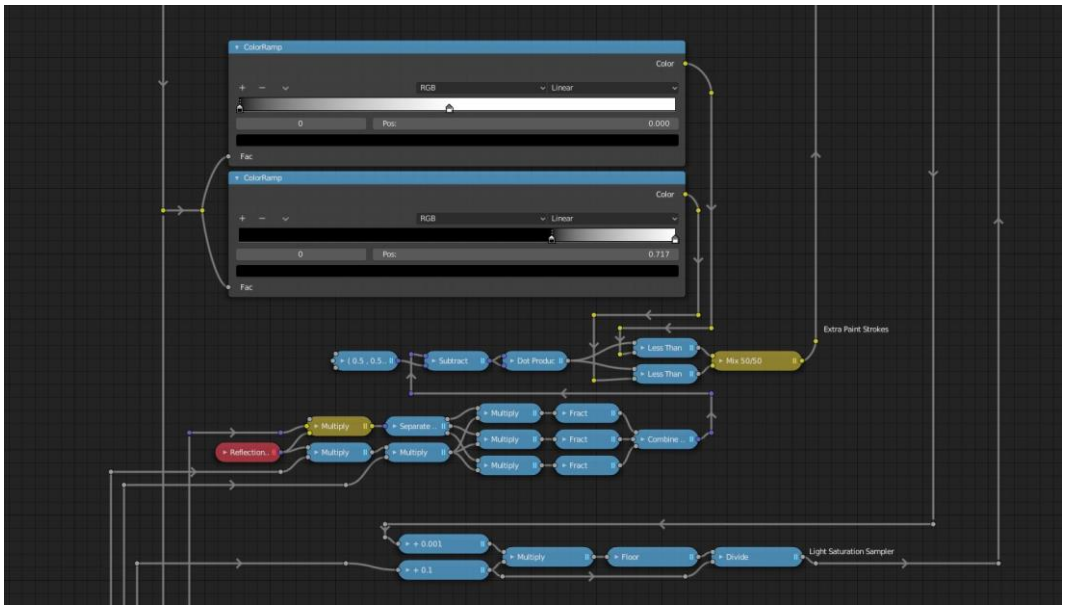


Figura 39: lo Shader Editor di Blender

Lo Shader Editor dispone di un sistema di programmazione basato su nodi, connettendo i diversi blocchi funzione si riesce ad implementare la logica desiderata. Le possibilità sono potenzialmente infinite, ma nel caso del progetto si è comunque limitati all'utilizzo di quanto è nativamente implementato: sebbene Blender permetta l'aggiunta di blocchi personalizzati tramite OSL[9], questa opzione non è correntemente disponibile per il motore di rendering Eeeve. Ciò nonostante, a partire da funzioni elementari è possibile tentare di ottenere risultati analoghi, anche se ciò implica quasi sicuramente un processo più scomodo rispetto alla stesura di qualche linea di codice.

[8] https://docs.blender.org/manual/en/latest/editors/shader_editor/index.html
[9] Open Shading Language: http://opensource.imageworks.com/

Brevemente, i nodi possono essere riassunti nelle seguenti categorie[10]:

- Input: raccoglie tutti gli stream di dati d'ingresso, incluse le *Texture Coordinates*, valori di *Fresnel*, informazioni della geometria a cui è applicato il materiale (normali, posizione, ecc.) e informazioni sui raggi di luce che interagiscono con l'oggetto (riflessione, rifrazione, diffusione, ecc.)
- Shader: sono tutti i nodi che implementano modelli fisici in grado di fornire autonomamente un output effettuando calcoli sulla luce in scena. Sono disponibili opzioni che spaziano dai più semplici *Diffuse* e *Glossy*, fino ad arrivare ai più complessi *Glass* e *Principled BSDF*.
- Texture: oltre ad includere i moduli che permettono di importare immagini personalizzate, raccoglie tutte le texture procedurali nativamente supportate da Blender.
- Color: nodi che permettono di operare sui canali colore, fra gli altri permettendo funzioni di mix e blend fra flussi di dati (overlay, burn, darken, soft light, ecc.). In caso di assenza di altre alternative, oltre ad operare su canali RGB, sono utili per eseguire operazioni fra vettori considerando XYZ come RGB. Questo è possibile grazie al fatto che Blender internamente tratta tutti i valori di colore linearmente, senza applicare curve di gamma e senza eseguire operazioni di clamp sui valori superiori a 1.0.
- Vector: include tutte le operazioni disponibili sui vettori, trasformazioni (nodo *Mapping*), cambi di sistemi di coordinate.
- Converter: raccolta di nodi che permettono di separare flussi di dati vettoriali nelle diverse componenti e viceversa. Fra questi figurano inoltre tre dei nodi più importanti e più utilizzati nel corso del progetto: *Color Ramp*, *Math* (operazioni e funzioni matematiche elementari), *Shader To RGB* (si veda

[10] https://docs.blender.org/manual/en/latest/render/eevee/materials/nodes_support.html

- REAL TIME VS path tracing).

3.3.3 Sistemi di coordinate per texture 3D

Prima di trattare i tipi di noise e pattern che verranno generati, è utile introdurre i diversi tipi di coordinate da cui si può partire per ottenere i risultati ricercati. In particolare, ci si soffermerà su quelli messi a disposizione nell'ambiente di shading a nodi di Blender.

Il nodo più importante a questo proposito è sicuramente *Texture Coordinate*, il quale dà accesso ai diversi sistemi di coordinate che verranno in seguito illustrati. Vi sono inoltre altri due nodi, Object Info e Camera Data, le quali permettono di eseguire alcune operazioni vettoriali interessanti se combinate agli altri input.

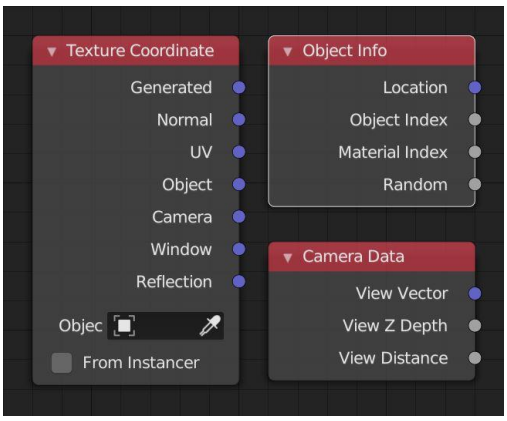


Figura 40: i nodi di input per le coordinate

Nella *Figura 41* è rappresentata la configurazione di test con la quale verranno illustrate le differenti coordinate. In particolare, da sinistra verso destra sono presenti:

- Una sfera di raggio 1
- Un cubo di dimensioni 1 x 1 x 1 e scala uniforme
- Un cubo di dimensioni 1 x 1 x 1 e scala 1 x 1 x 2
- Un parallelepipedo di dimensioni 1 x 1 x 2 e scala uniforme

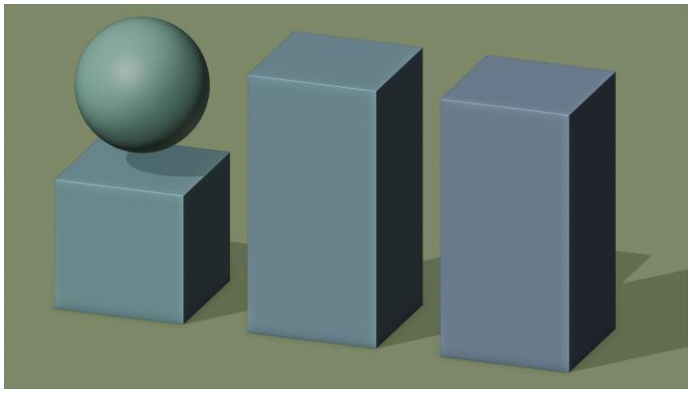


Figura 41

Si presti particolare attenzione alle differenze che emergeranno fra il cubo scalato lungo l'asse z (nella modalità *Object Mode* di Blender) e il parallelepipedo di eguali dimensioni ma con scala uniforme (scalato in *Edit Mode*).

Le coordinate saranno rappresentate con dei colori, interpretando i valori di XYZ come RGB. La griglia del pavimento rappresenta le coordinate globali, ogni quadrato ha dimensione 1 x 1.
Di seguito degli esempi per chiarire come si possano leggere le coordinate dalle figure che seguiranno:

- **(0, 0, 0)** colore nero (origine)
- **(1, 1, 1)** colore bianco
- **(1, 0, 1)** colore viola
- **(0, 0, 1)** colore blu

UV

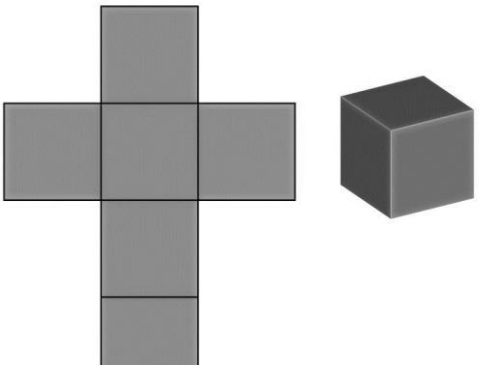


Figura 42: : UV unwrap di un cubo

Le coordinate di tipo *UV* sono frequentemente utilizzate quando si desidera mappare immagini su superfici dove è necessario avere un controllo preciso da parte dell'artista. Con questa modalità è infatti possibile definire una corrispondenza diretta fra un punto della superficie e

una sua rappresentazione nello spazio 2D. Il processo di definizione di questa relazione è detto *UV unwrap* ed è solitamente manuale. Quest'ultima caratteristica rende quindi inadatto allo scopo questo tipo di proiezione, essendo l'obiettivo quello di minimizzare l'intervento manuale. Ciò non toglie che alcune texture saranno realizzate appositamente e necessiteranno di questo tipo di operazione (per esempio nei personaggi).

Generated

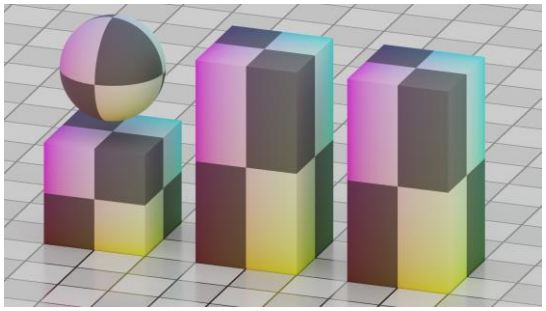


Figura 43: coordinate Generated

Le coordinate di tipo *Generated* sono quelle utilizzate automaticamente da Blender nel caso in cui venga applicata una texture ad un modello che non possiede una mappatura UV. Il modo in cui lo spazio è generato è il seguente: viene considerata la *bounding-box*[11] dell'oggetto, lungo le tre direzioni della stessa vengono definiti i tre versori delle coordinate, i valori delle coordinate sono quindi mappati da 0 a 1 lungo i bordi della bounding-box. Il risultato è che la distorsione dello spazio è altamente dipendente dalla simmetria della bounding-box: più questa è vicina ad un cubo, minore sarà la distorsione. Si noti come nella *Figura 43* i due parallelepipedi presentano lo stesso risultato, proprio perché l'unico fattore ad essere rilevante è la dimensione finale dell'oggetto.

Discorso a parte è quello della rotazione, il sistema è comunque solidale all'oggetto, rotandolo quindi in Object Mode anche le coordinate ruoteranno. Se l'oggetto è ruotato in Edit Mode, questa trasformazione avverrà precedentemente al calcolo della bounding-box, con il solo risultato di cambiarne, probabilmente, le proporzioni.

[11] Il più piccolo parallelepipedo in grado di inglobare completamente l'oggetto.

In generale l'utilizzo dell'input di tipo Global è buono quando si vuole mappare un'immagine sapendo che questa verrà stirata sull'interezza dell'oggetto. Nel caso del progetto, questo genere di distorsioni non è ottimale, sarà quindi evitato l'utilizzo il più possibile. Quando sarà fatto si cercherà di compensare la distorsione successivamente con delle trasformazioni, oppure sarà per situazioni in cui la distorsione non è percettibile.

Object (local)

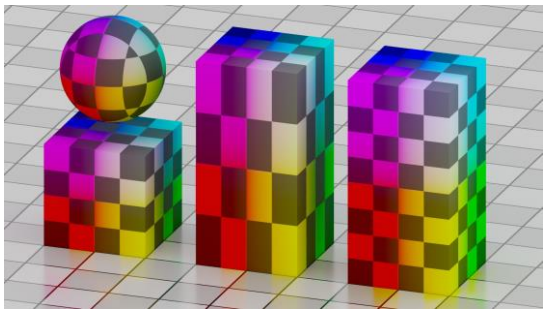


Figura 44: coordinate Local

Molto simili al tipo sopra trattato, generano uno spazio partendo dall'origine dell'oggetto. A differenza delle precedenti però, queste sono indipendenti dalla dimensione dell'oggetto, si estendono al contrario in modo non distorto lungo le tre direzioni. Si potrebbero quindi definire coordinate di tipo locale.

Si noti come dalla figura emerga l'influenza del fattore di scala del primo parallelepipedo: una volta generato lo spazio, questo è infine sottoposto alle trasformazioni applicate all'oggetto. Si nota in particolare come il secondo parallelepipedo, pur avendo la stessa dimensione finale, non presenta distorsioni della griglia.

Nel caso di studio il sistema è dunque preferibile a quello Global.

Si noti però come questo genere di coordinate sia generato successivamente all'applicazione dei modificatori, potendo quindi generare in alcuni casi degli artefatti, per esempio in caso di applicazione di deformazioni con armatura.

Position (global)

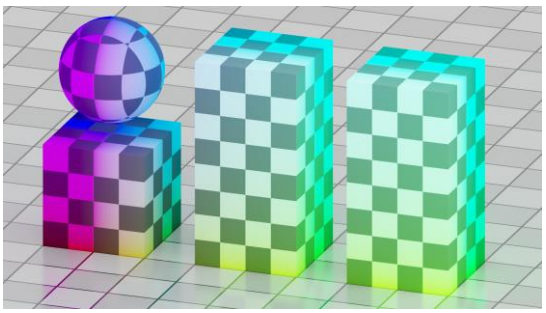


Figura 45: coordinate Position

Le coordinate di tipo *Position* sono solitamente definite globali negli altri software, la loro particolarità è che ad ogni punto di una superficie viene associato un valore pari alla sua posizione nello spazio. Questo metodo è particolarmente interessante perché permette di realizzare texture 3D che non dipendono dalle trasformazioni di scala e rotazione degli oggetti in scena, ma solamente dalla loro posizione finale. È quindi particolarmente utile quando viene usate su oggetti posizionati rapidamente e in maniera casuale, in questo modo non è necessario prendersi cura della scala della texture in quanto essa è uniforme su tutto lo spazio. Nella *Figura 45* si può notare come questo comportamento sia rispecchiato dal fatto che, nonostante dimensioni e posizioni degli oggetti siano differenti, il gradiente si estende in modo continuo attraverso questi ultimi.

Normal

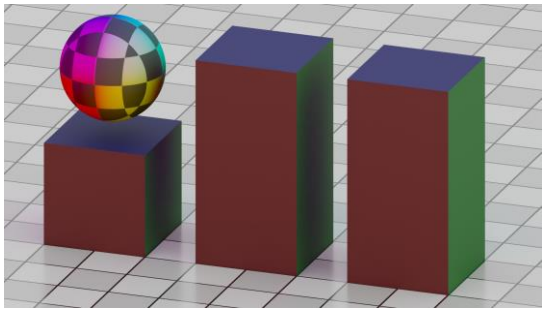


Figura 46: coordinate Normal

Le coordinate di tipo *Normal* definiscono uno spazio in cui ad ogni punto sulla superficie di un oggetto è associato un versore normale ad essa. Possono essere molto utili nel caso in cui si vogliano introdurre degli effetti solo quando l'oggetto è orientato lungo una certa direzione.

Manipolandone la direzione con delle funzioni apposite e collegandole ad uno shader è inoltre possibile generare delle variazioni interessanti sulla luce

che cade sull'oggetto (sarà affrontato in seguito per quanto concerne la stilizzazione della *rim light*).

Camera

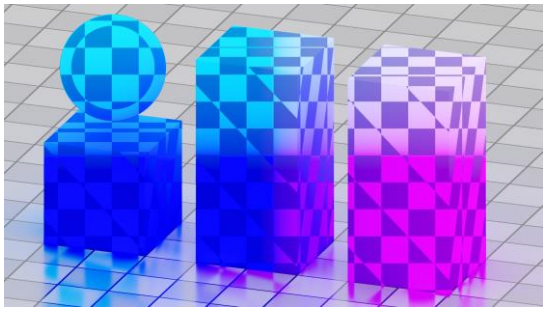


Figura 47: coordinate Camera

Le coordinate di tipo Camera sono sicuramente fra le più interessanti perché aprono a diverse opzioni creative sfruttando le possibilità matematiche che ne derivano. In particolare, ad ogni punto della superficie viene associato il vettore con direzione e verso che unisce la posizione della camera al punto stesso sulla superficie, il modulo è pari alla distanza.

Come si può notare dalla figura, utilizzare questi valori per generare direttamente una texture può portare a risultati inaspettati e privi di senso. Questo proprio perché, come detto precedentemente, questi dati sono utili per eseguire operazioni e non per un utilizzo diretto.

View Vector

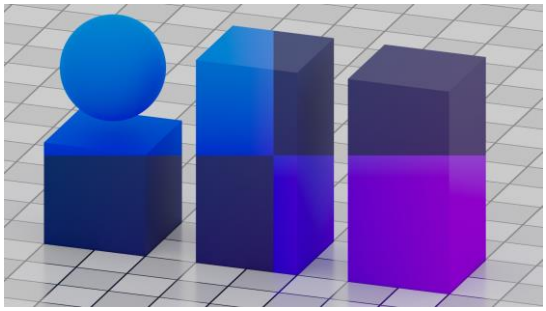


Figura 48: coordinate View Vector

Le coordinate *View Vector* sono un parente stretto del tipo Camera, anch'esse danno informazioni sul vettore che unisce punti della superficie alla camera, la differenza sta nel fatto che questi vettori sono normalizzati.

Avendo una componente di profondità costante, i dati forniti possono essere utilizzati in modo più agevole rispetto a quelli di tipo Camera.

Si dimostrerà in seguito come View Vector porti con sé informazioni legate alla distorsione prospettica dovuta alle diverse lunghezze focali, e come questo possa essere utile alla generazione di pattern interessanti.

Window (screen)

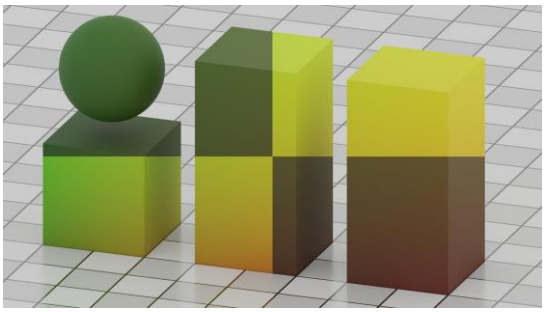


Figura 49: coordinate Window

Le coordinate di tipo *Window* corrispondono ad una versione normalizzata di quelle che solitamente si definirebbero coordinate di tipo screen space.

In particolare, vengono definiti due assi ortogonali corrispondenti all'orizzontale e al verticale dell'immagine impressa sul sensore virtuale della camera in scena.

Il fatto che le coordinate siano normalizzate rende necessario considerare il rapporto d'aspetto (aspect ratio) per compensare il fattore di stiramento introdotto. Una volta risolto questo difetto intrinseco, il risultato è uno spazio estremamente versatile e utile alla realizzazione di diversi effetti in screen space. Si vedrà in seguito come queste coordinate in particolare si riveleranno essere particolarmente utili allo scopo del progetto.

Reflection

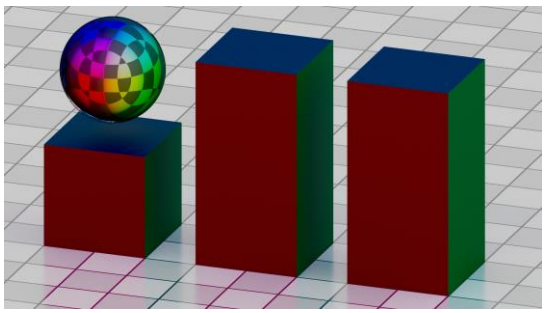


Figura 50: coordinate Reflection

Tra le più particolari di quelle disponibili, le coordinate Reflection usano restituiscono la direzione del versore di riflessione per ogni punto della superficie. Possono essere utilizzate per aggiungere riflessioni fasulle agli oggetti, non troveranno però grande applicazione nel progetto.

Tangent

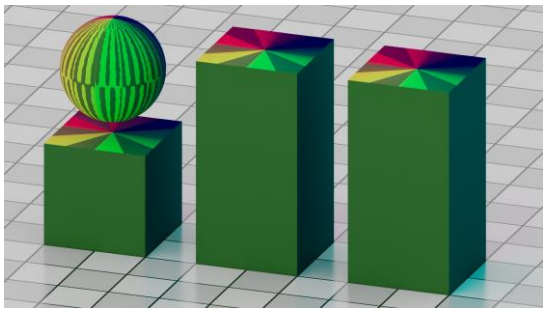


Figura 51: coordinate Tangent

Concettualmente simili a Normal, le coordinate Tangent restituiscono il campo dei vettori tangenti alla superficie dell'oggetto. Possono essere utilizzate per simulare dei comportamenti anisotropi o creare variazioni nelle texture in relazione alla curvatura del modello.

3.3.4 Texture 3D

Una volta analizzati i diversi spazi vettoriali di cui si dispone, è possibile considerare ciò che Blender può nativamente generare a partire da essi. Si passeranno dunque ora in rassegna i principali nodi di tipo Texture. Per semplificazione si possono distinguere due macrocategorie (anche se le distinzioni fra i due presentano talvolta delle sfumature): *pattern* e *noise* (rumori).

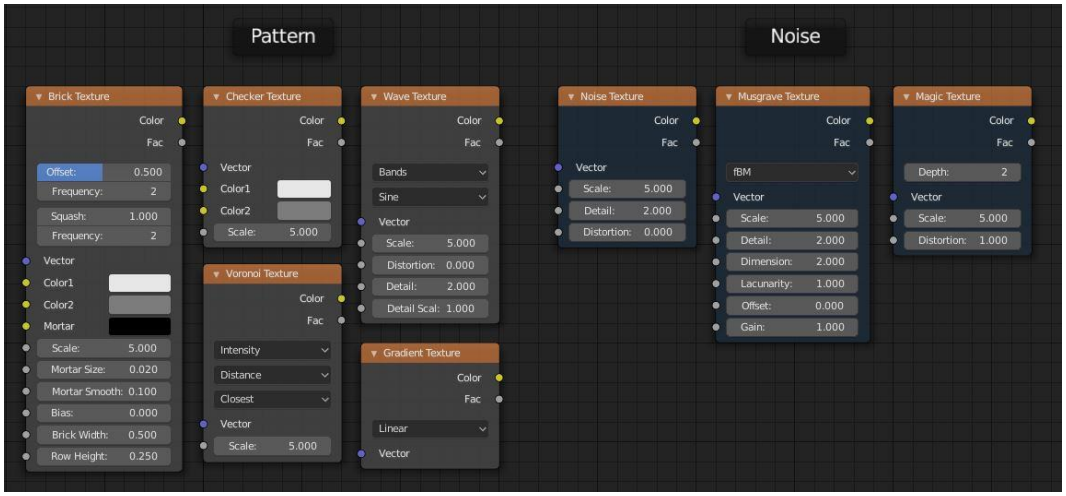


Figura 52: nodi texture

Nella *Figura 52* si può notare come i parametri a disposizione siano molti e vari. Ciò che accomuna tutti è l'input di tipo Vector, il quale riceve le informazioni riguardo le coordinate da cui partire per ottenere i valori finali. Le texture 3D infatti non sono altro che funzioni matematiche che ricevono in input le coordinate di un punto e restituiscono un valore che può essere in scala di grigi o colorato.

La trattazione approfondita delle caratteristiche di ognuna delle funzioni è decisamente qualcosa di molto vasto e non necessario in questa sede, ci si limiterà dunque ad esaminare solo gli elementi più interessanti ai fini del progetto.

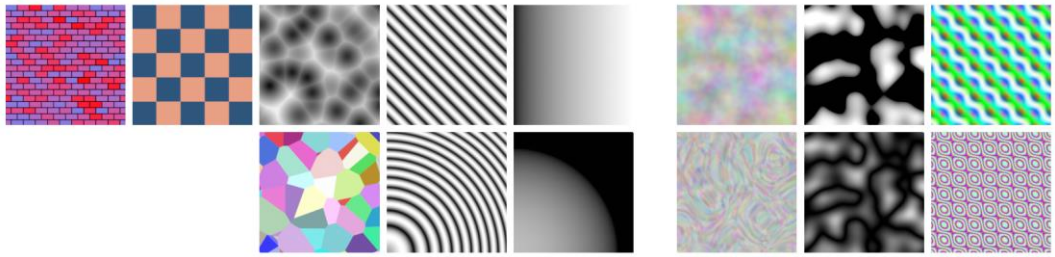


Figura 53: alcuni pattern (sinistra) e noise (destra) ottenibili.
In ordine da sinistra verso destra: brick, checker, Voronoi, wave, gradient, Perlin, Musgrave e magic.
Al di sotto di alcuni sono presenti delle variazioni di output disponibili.

Pattern

I pattern disponibili internamente possono essere riassunti con il seguente elenco.

- *Brick*, utile per realizzare mattoni e in generale forme con suddivisione a griglia con elementi a colori casuali. Si possono variare scala, offset dei mattoni e scala della linea di suddivisione (mortar).
- *Checker*, genera scacchiere 3D alternando fra due colori specificati dall'utente.
- *Voronoi*, genera delle celle basate su un campo di distanza fra punti distribuiti in modo casuale nello spazio. Verrà analizzato in modo approfondito in seguito.
- *Wave*, implementa funzioni periodiche di tipo seno e dente di sega, è anche disponibile in versione ad anelli.
- *Gradient*, genera un gradiente con valori da 0 a 1 secondo il metodo specificato dall'utente.

Il più interessante dell'elenco, come preannunciato, è sicuramente il pattern Voronoi, sia per funzionalità implementate che per l'algoritmo stesso che lo genera. I principi dietro all'algoritmo saranno infatti utilizzati in seguito per realizzare un'implementazione personalizzata dello stesso. Si ritiene quindi importante farne un'analisi più approfondita.

Diagramma di Voronoi

In matematica, un diagramma di Voronoi (dal nome di Georgij Voronoi), anche detto tassellatura di Voronoi, è un particolare tipo di decomposizione di uno spazio metrico determinata dalle distanze rispetto ad un determinato insieme discreto di elementi dello spazio (ad esempio, un insieme finito di punti).[12]

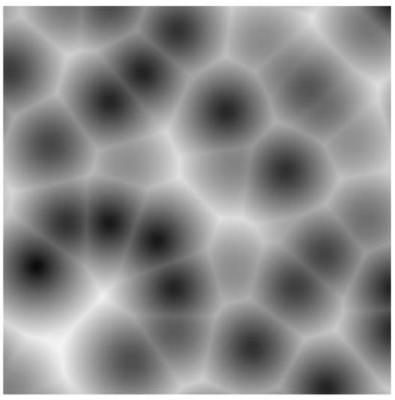


Figura 54: campo di distanza generato dalla partizione di Voronoi

Il motivo per cui è particolarmente rilevante nel caso di studio è la sua capacità di generare forme poligonali a partire da alcuni punti definiti. Dati un set di punti S è infatti possibile calcolare un campo di distanze tali per cui per ogni altro punto nello spazio è associata la distanza dal più vicino punto appartenente ad S. Si noti come nella figura a lato si vadano a definire delle linee di delimitazione fra celle dove l'intensità del campo è maggiore.

È successivamente possibile eseguire delle operazioni di clamp sul campo per ottenere forme e pattern più complessi. Va evidenziato che, essendo una distanza euclidea, con questo di operazioni si otterranno sempre delle forme circolari. Blender implementa altri metodi per il calcolo delle distanze (Taxicab, Chebyshev e il più generico Minkowski), oltre alla possibilità di specificare se considerare durante il calcolo il punto più vicino, il secondo e così via[13].

[12] https://it.wikipedia.org/wiki/Diagramma_di_Voronoi
[13] https://docs.blender.org/manual/en/latest/render/shader_nodes/textures/voronoi.html

Noise

Analogamente a quanto fatto con i pattern, possiamo qui riassumere i rumori disponibili all'interno del contesto nodi di Blender:

- *Perlin*, internamente chiamato "noise", è il rumore standard nella computer grafica, dà un risultato simile a delle nuvole, verrà approfondito di seguito.
- *Musgrave*, è un rumore di tipo frattale che genera diverse iterazioni di rumore Perlin unendole secondo il metodo specificato.
- *Magic*, genera variazioni di colore dall'aspetto psichedelico, oltre alla capacità di generare pattern di punti se opportunamente trattato è praticamente inutile nel caso di studio.

Anche in questo caso ci si soffermerà sulla funzione più interessante e utile, in particolare il rumore di tipo Perlin.

Perlin

Il rumore di Perlin è un tipo di rumore descritto nel 1983 dall'informatico statunitense Ken Perlin.[14] L'obiettivo era quello di generare un noise in grado di essere utilizzato in ambito di computer grafica per generare texture organiche dall'apparenza naturale.

Figura 55: rumore di Perlin

L'algoritmo di base funziona partendo da una griglia di punti generata in uno spazio *n*-dimensionale. Per ognuno dei punti viene generato un vettore *n*-dimensionale casuale. Partendo da questi valori casuali viene poi effettuata un'interpolazione in modo da "riempire" lo spazio rimanente. Così facendo si ottiene una variazione casuale distribuita nello spazio.

[14] https://en.wikipedia.org/wiki/Perlin_noise

Osservando la *Figura 55* si nota subito come il rumore così ottenuto soffra di mancanza di dettagli. Questo è facilmente risolvibile introducendo il concetto di ottava (in Blender contrassegnato dal valore *Detail*).
È infatti possibile scalare della metà il noise ottenuto al passaggio precedente e sommarlo con ampiezza dimezzata alla versione non scalata. In questo modo si ottiene un dettaglio maggiore. Il procedimento ha natura frattale e può essere iterato più volte, ottenendo così la quantità desiderata di dettagli.

Figura 56: rumore di Perlin con 0, 1 e 16 ottave

Blender dà inoltre la possibilità di poter operare su un parametro di distorsione, ottenendo così ulteriori variazioni.

Oltre ad essere un potente strumento per la generazione di texture in sé, nel caso del progetto il rumore sarà più che altro considerato per eseguire delle operazioni di distorsione sulle coordinate utilizzate per generare altri pattern. Il principio di manipolazione tramite somma, mix o moltiplicazione dei noise al fine di utilizzarne i valori come coordinate è forse uno degli strumenti più potenti di cui si dispone in ambito di texture procedurali.

4 *Light Studio* add-on

Nel corso della produzione, prima di procedere con la vera e propria realizzazione dei contenuti negli ambienti 3D, ci si è chiesti in che modo il team potesse investire le proprie risorse umane per migliorare programmaticamente il flusso di lavoro. Avendo scelto Blender come software principale per seguire l'intera pipeline di produzione, si sono analizzati i punti deboli e le criticità.

Uno dei punti sui quali si è pensato di intervenire è stato quello di una ottimizzazione dell'interfaccia per la gestione delle luci di scena, un'ottimizzazione che, auspicabilmente, permettesse di risparmiare tempo, dato le esigue risorse a disposizione.

Come già anticipato nel paragrafo "Blender API", il primo passo per ottenere il risultato desiderato è stato lo studio della documentazione, focalizzando l'attenzione sugli argomenti di maggiore interesse per l'obiettivo da perseguire. Successivamente si è allestita l'infrastruttura per lo sviluppo.
Nella fattispecie, si è deciso di sviluppare il plugin Light Studio in Visual Studio Code, versione "light" del più famoso Visual Studio, ambiente di sviluppo di Microsoft nella sua versione 1.39.2.[15]
La suddetta configurazione ha così permesso di sviluppare il codice in un vero e proprio ambiente di sviluppo (con tutti i benefici che questo comporta) potendo eseguire il debug sia all'interno dell'ambiente stesso sia in Blender. Visual Studio Code offre un add-on che permette di testare in tempo reale il codice scritto, direttamente in Blender andando così a snellire notevolmente il processo di bug fix e il test di piccole modifiche apportate.

[15] https://code.visualstudio.com/

4.1 UX design

Nel seguente paragrafo ci si pone come obiettivo quello di confrontare i principali cambiamenti apportati all'interfaccia di Blender con l'introduzione di Light Studio.

4.1.1 Interfaccia nativa Blender

Usando Blender per l'allestimento degli ambienti e per il master lighting ci si è resi conto che l'accesso alle proprietà delle luci risultava un po' macchinoso: solamente dopo la selezione di una luce, è possibile monitorare ed intervenire sulle sue proprietà. Inoltre, questo è possibile esclusivamente dal *Context: Object data* del pannello *Properties*(*Figura 57*).

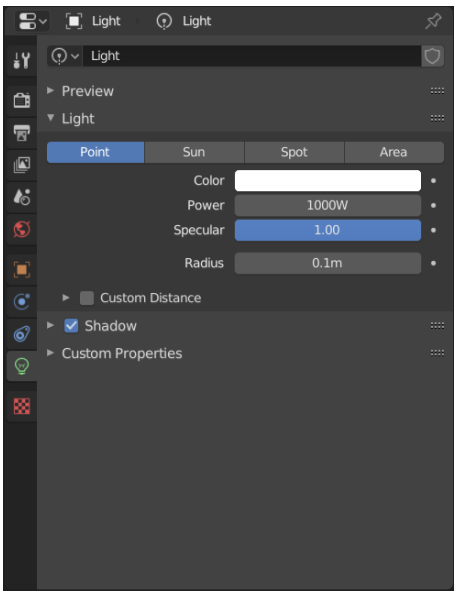


Figura 58: context: object data di una luce

Questo comporta una serie di limitazioni:

- È sempre necessario selezionare la luce per poterne modificare gli attributi.
- Avendo selezionato una determinata luce in scena, non è possibile monitorare (e ancor meno modificare) gli attributi di un'altra luce che concorre all'illuminazione della stessa scena.

Risulta chiaro dunque, che per scene complesse in cui sono presenti più luci risulti tutto esponenzialmente più macchinoso e richieda molto tempo anche per poter testare piccoli cambiamenti.
Proprio in questo contesto viene concepito Light Studio che si pone come obiettivo quello di fornire uno spazio dedicato all'interno dell'interfaccia di Blender che permetta di monitorare ed accedere alle proprietà di tutte le luci contemporaneamente, senza badare all'oggetto selezionato.

4.1.2 Interfaccia Light Studio

Per il plugin Light Studio si è deciso di sfruttare lo spazio che Blender mette a disposizione di default per i nuovi plugin: il menu contestuale *Settings* all'interno dello spazio *View 3D*.

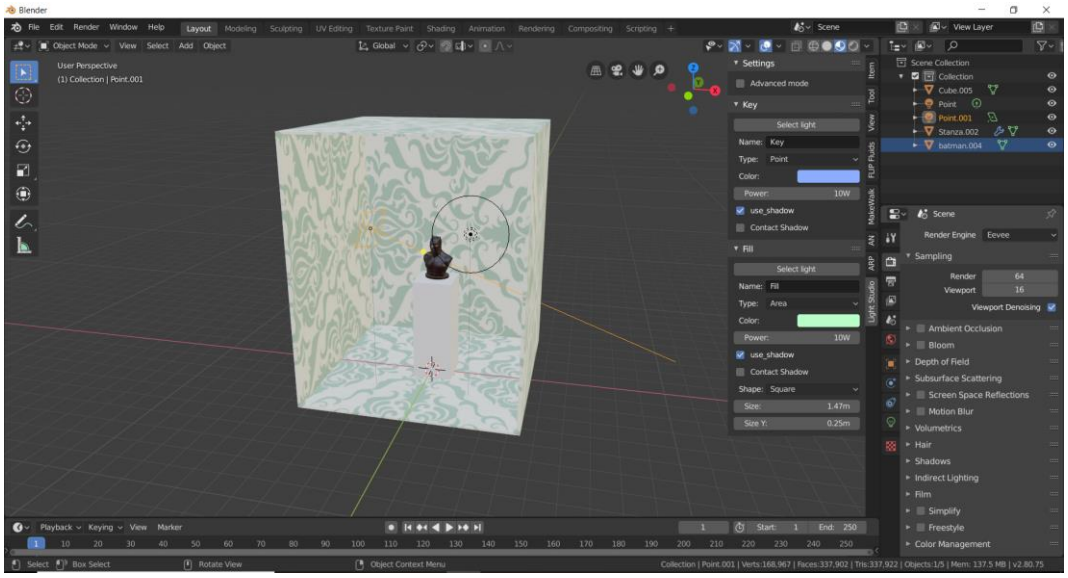
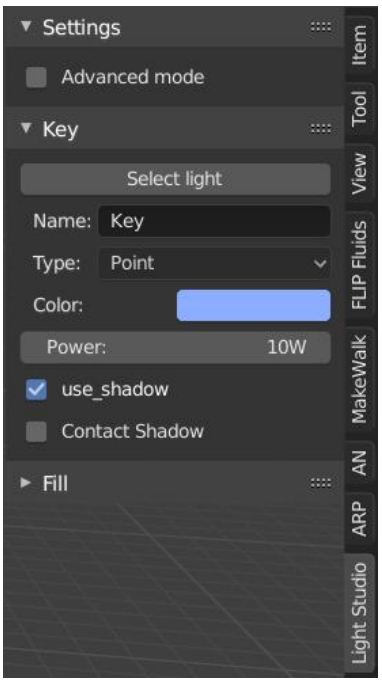


Figura 59: schermata di Blender con Light Studio

Così facendo, è possibile sfruttare l'intera altezza della finestra dedicata alla vista 3D per monitorare il maggior numero possibile di parametri relativi alle luci in scena. Un contesto del genere ha permesso dunque, di creare una "istanza grafica" (menu con gli attributi di una singola luce) per ciascuna luce. Queste istanze possono essere ridotte ad una singola riga (per poter risparmiare spazio nel caso in cui si sia soddisfatti del risultato ottenuto o semplicemente per un'ottimizzazione del viewport) e riordinate manualmente dall'alto verso il basso con una semplice operazione di *drag and drop*, andando a creare una gerarchia per poter introdurre eventuali cambiamenti a partire dalle luci principali e proseguire con quelle di minore importanza.

Sempre nell'ottica di una gestione più oculata dello spazio all'interno dell'interfaccia, si è deciso di introdurre una doppia modalità di visualizzazione delle proprietà delle luci:

- Base: sono visibili e modificabili solamente le proprietà base delle luci. Si tratta di una modalità che richiede poco spazio all'interno dell'interfaccia ma che espone le proprietà per le modifiche principali da apportare alle luci in scena.
- Advanced: sono esposte un gran numero di variabili (la quasi totalità delle proprietà aggiunte all'interfaccia riguarda la gestione delle ombre e delle ombre di contratto). Ciascuna luce occuperà una consistente parte dell'interfaccia ma sarà possibile un controllo estremamente raffinato di tutte le proprietà. In ogni caso, non bisogna dimenticare che il pannello dedicato a ciascuna luce può essere ridotto ad una sola riga per riservare più spazio alle luci di nostro interesse e con lo scorrimento verticale sarà comunque possibile raggiungere le proprietà di tutte le luci.



È possibile cambiare modalità in qualunque momento andando a validare (o svuotare, a seconda delle situazioni) la casella di controllo *Advanced mode* presente nella parte superiore del pannello di Light Studio, nella categoria *Settings*.

Il vero valore aggiunto di una interfaccia costruita in questo modo è che è possibile operare senza dover necessariamente selezionare la luce (ed il relativo pannello Object data). Inoltre, anche in caso di modalità Advanced ed un gran numero di luci in scena, è possibile navigare all'interno del plugin con un semplice scorrimento verticale.

4.2 Codice

Come già accennato in precedenza, Light Studio nasce con l'obiettivo di ottimizzare dal punto di vista grafico l'interfaccia di Blender. Per poter garantire

la massima efficienza è stato fondamentale che il plugin mostrasse nel proprio menu sempre dati aggiornati rispetto a quanto accade in scena.

Dopo una prima analisi, si sono presentate due possibilità:

- Aggiornare l'elenco delle luci "brute force" ogni qual volta Blender lancia un evento.
- Gestire in maniera separata i diversi eventi che posso occorrere, ed aggiornare in maniera adeguata l'elenco di luci.

A causa del gran numero di eventi lanciati da Blender, l'overhead computazionale è significativo, specialmente quando il numero di luci presenti nella scena cresce. Per questo si è deciso di adottare la seconda soluzione.
Per poter gestire gli eventi significativi si è deciso di racchiudere la maggior parte della logica del plugin all'interno della classe `LightStudioPlugin` definita nel seguente modo.

```
Class LightStudioPlugin:


        def __init__(self):
        self.managed_lights = {}
        self.advanced_mode = False
        self.advanced_properties_ids = set([…])
        self.hidden_properties_ids = set([…])
```

Il dizionario `managed_lights` è il cuore del plugin per quanto riguarda la gestione dei dati. Gli oggetti presenti al suo interno sono delle `managedLight` identificate dalla stringa che viene usata come nome della luce in Blender (chiave).
Data la strada intrapresa, diventa fondamentale avere un mapping perfetto tra le luci effettivamente in scena e le luci presenti all'interno del dizionario `managed_lights`.
Per poterlo fare, è stato definito il metodo `refresh_managed_lights(self)`

che permette di gestire in maniera ottimale una serie di casi limite che avrebbero potuto compromettere la sincronia tra le luci in scena e le luci nel dizionario `managed_lights`:

- Undo: per poter permettere agli utenti di tornare indietro nella cronologia delle azioni effettuate, Blender continua a tenere in memoria (per un certo intervallo di azioni) gli oggetti eliminati in una lista parallela a quella degli oggetti in scena. Nel caso in cui alcune luci vengano ripristinate dopo l'eliminazione, si sarebbero create delle ambiguità che se non gestite correttamente, avrebbero potuto mettere a repentaglio la correttezza delle informazioni mostrate nel plugin.
- Rinomina luci: avendo scelto di usare come chiave per il dizionario `managed_lights` il nome della luce, in caso di rinomina sarebbero stati creati dei duplicati riferiti alla stessa luce.

```
def refresh_managed_lights(self):
        if not hasattr(bpy.data, 'lights'):
            return


        scene_objects = [o.data for o in bpy.context.scene.ob-
jects.values()]
        new_lights = [l.data for l in bpy.context.scene.ob-
jects.values() if isinstance(l.data, bpy.types.Light)]
        for light in new_lights:
            if light.name in self.managed_lights:
                if self.managed_lights[light.name].light_id ==
id(light):
                    continue
                unregister_class(self.mana-
ged_lights[light.name].light_panel)
            plugin_light = ManagedLight(light)
```

```
            self.managed_lights[light.name] = plugin_light
            register_class(plugin_light.light_panel)


        obsolete_lights = {k: v.light for k, v in self.mana-
ged_lights.items() if v.light not in scene_objects}
        for light_key, light in obsolete_lights.items():
            unregister_class(self.mana-
ged_lights[light_key].light_panel)
            del self.managed_lights[light_key]


        renamed_lights = [l for l in self.mana-
ged_lights.keys() if self.managed_lights[l].light.name != l]
        for l in renamed_lights:
            unregister_class(self.managed_lights[l].light_pa-
nel)
            del self.managed_lights[l]
```

Avendo definito il modo in cui le luci vengono collezionate all'interno del plugin, è necessario estrarre le proprietà da esporre.
Trattandosi delle proprietà che Blender mette a disposizione nella propria interfaccia nativa, è bastato estrarle dall'RNA. In questo modo, oltre evitare di ridefinire programmaticamente tutte le proprietà, viene garantito automaticamente il binding tra le proprietà nel pannello *Object data* e nel pannello di *Light Studio*.

4.3 Ottimizzazione produzione

Per poter verificare l'efficacia di Light Studio sono stati realizzati dei test da sottoporre ad utenti di Blender con diversi livelli di esperienza e nel presente paragrafo verranno riportati i risultati ottenuti.

Sono stati formulati due task e si è chiesto agli utenti di portarli a termine una prima volta usando esclusivamente l'interfaccia nativa di Blender. Successivamente si è chiesto di ripetere il task, usando l'interfaccia messa a disposizione da Light Studio.

I due task da portare a termine sono stati i seguenti:

- Cambiare tipologia di luce da Point a Sun con intensità 5 ed impostare il colore a rosso (RGB: (255,0,0)).
- Creare una illuminazione a tre punti (rinominando adeguatamente le tre luci) con 3 Point light in cui la key light ha intensità doppia delle altre 2.

Entrambi i task sono stati svolti da una popolazione statistica di 30 individui a partire da un file campione comune.

4.3.1 Task #1

I risultati ottenuti sono stati eccellenti. Il tempo impiegato per portare a termine il task è stato notevolmente inferiore nella totalità dei casi. Come mostrato nella *Figura 60: confronto tempo impiegato per il completamento del task #1* sono stati misurati crolli del tempo impiegato in media del 36% con picchi del 61%.

Inoltre, nella maggior parte dei casi, gli utenti non hanno riportato alcuna inadeguatezza di *Light Studio* rispetto al task da portare a termine.

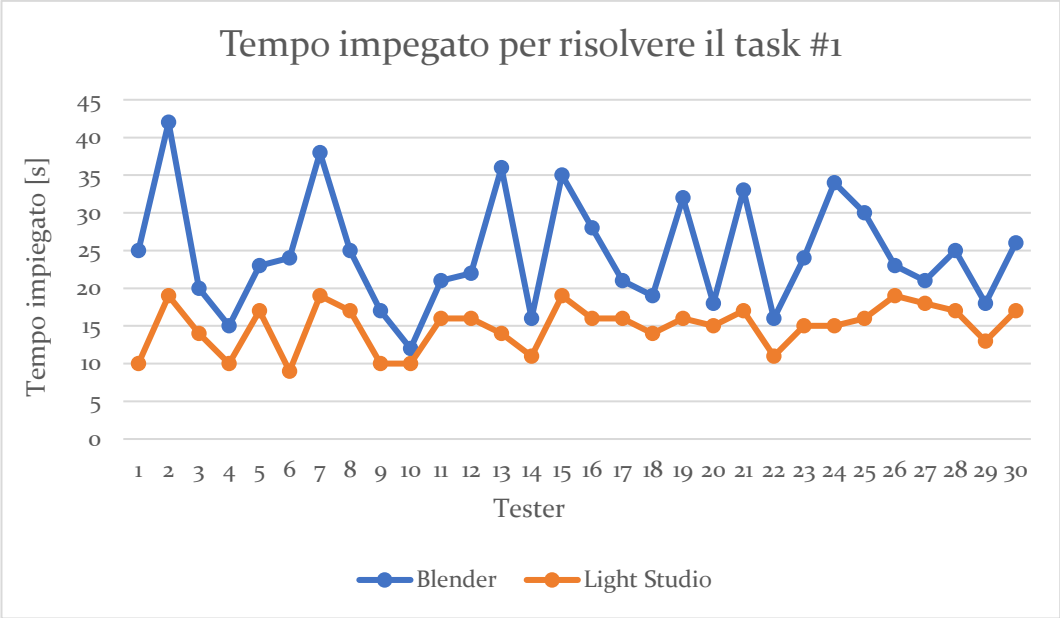


Figura 60: confronto tempo impiegato per il completamento del task #1

4.3.2 Task #2

Il secondo task è stato fondamentale per il processo di sviluppo.
Nella maggior parte dei casi il tempo impiegato con Light Studio è stato inferiore. Tuttavia, nei casi in cui il task è stato portato a termine in minor tempo usando la sola interfaccia di Blender, è emersa una criticità di Light Studio riscontrata da buona parte della popolazione di test.
L'apparente vantaggio di poter modificare le luci in scena senza la necessità di selezionarle si è rivelato un'arma a doppio taglio: muovendosi all'interno della scena senza oggetti selezionati, gli utenti si sono sentiti disorientati ed hanno faticato a capire su quale oggetto stessero effettivamente operando, in alcuni casi anche dopo avere rinominato le luci. Questo, dove occorso, ha portato ad un significativo ritardo nel compimento del task.

A seguito di un così serio riscontro si è deciso di rimettere mano al codice ed aggiungere all'interno dell'interfaccia di Light studio la possibilità di selezionare le varie luci sulle quali si può operare tramite un bottone.

In ogni caso, come già accennato in precedenza, i risultati ottenuti si sono rivelati particolarmente interessanti: la maggior parte della popolazione

(80%) ha risolto il task #2 in minor tempo con l'interfaccia di Light Studio. Considerando tutti i risultati ottenuti il tempo di svolgimento è stato abbattuto mediamente del 26% con picchi del 62%. Nei casi in cui il task #2 è stato risolto più efficientemente con l'interfaccia nativa di Blender (20% dei tester), sono stati riportati rallentamenti tra il 12% ed il 70%.

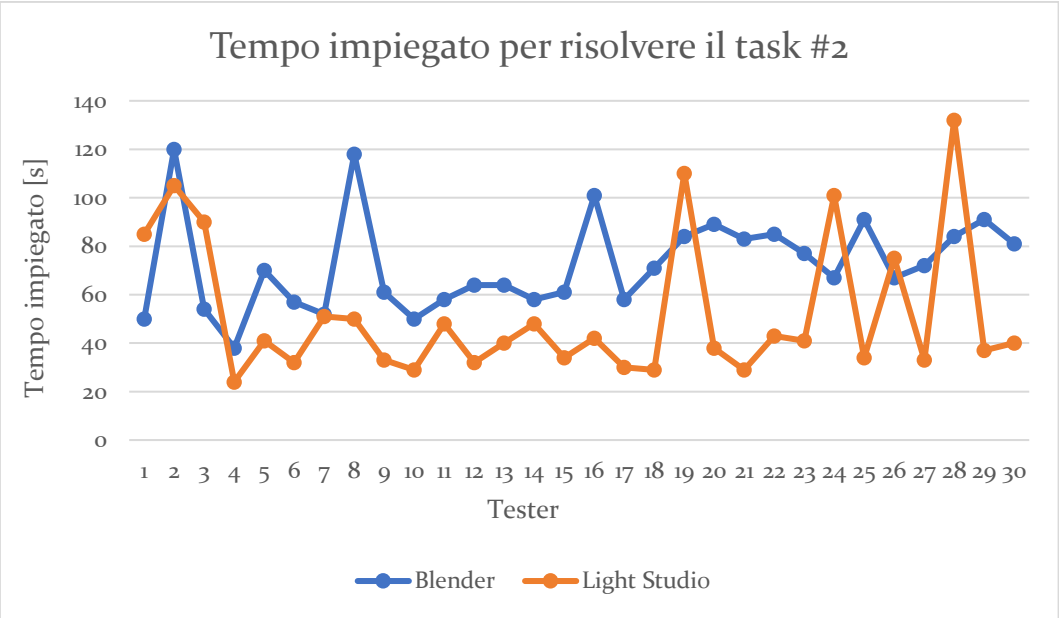


Figura 61: confronto tempo impiegato per il completamento del task #2

Poiché il task #2 richiedeva una maggiore interazione con il software rispetto al precedente, si è pensato che potesse essere utile raccogliere altre metriche per valutare l'efficienza del plugin. Si è deciso, dunque, di misurare il numero di click effettuati con il mouse ed il numero di tasti premuti sulla tastiera per potare a compimento il task #2 ed i risultati sono riportati rispettivamente nel *Figura 62* e nel *Figura 63*.

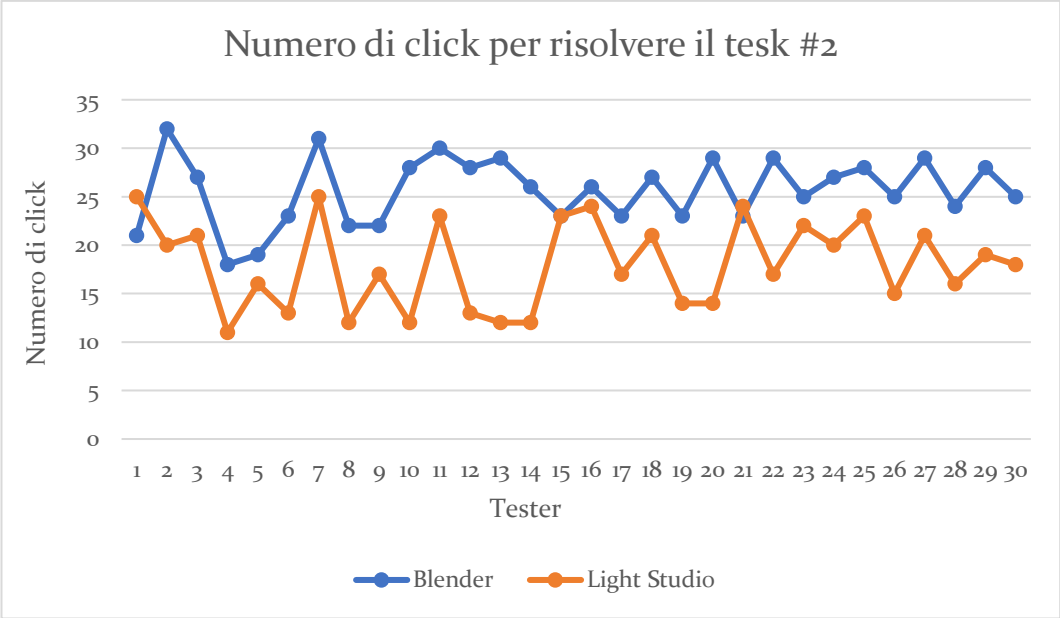


Figura 62: confronto numero di click impiegati per il completamento del task #2

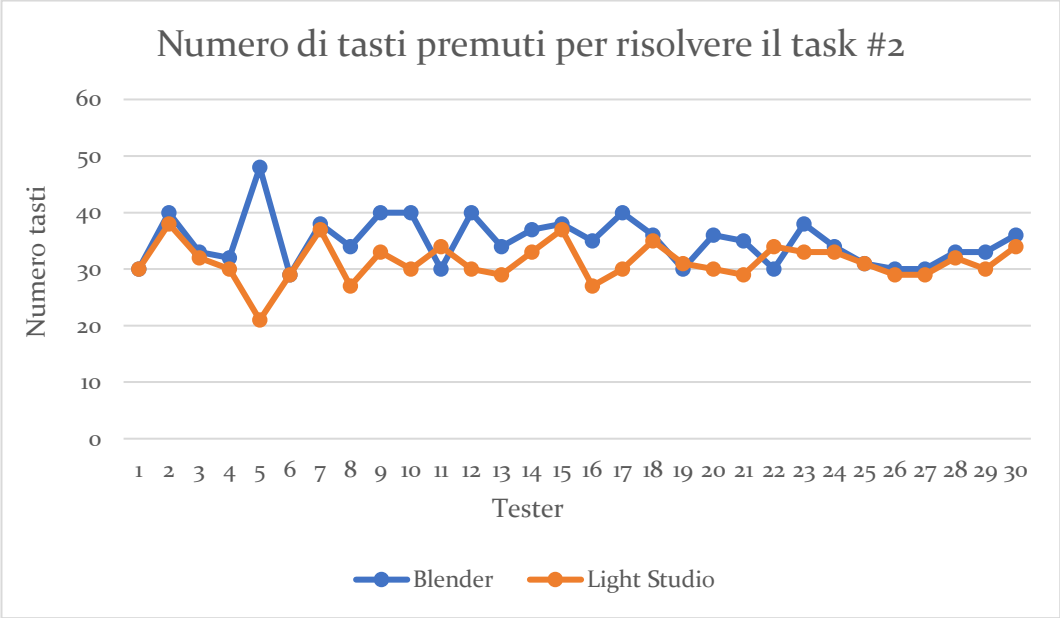


Figura 63: confronto numero di tasti premuti per il completamento del task #2

Nell'analisi dei grafici precedenti c'è una considerazione preliminare da fare: la varianza dei valori ottenuti dipende non solo dal grado di esperienza degli utenti che si sono sottoposti al test (come nel caso della metrica "tempo impiegato"), ma anche dal tipo di approccio. Alcuni utenti prediligono l'uso del mouse, altri preferiscono impartire comandi tramite le equivalenti scorciatoie da tastiera. Quindi i dati più consistenti che possono essere estrapolati

riguardano esclusivamente la riduzione percentuale del numero di click e numero di tasti premuti e non il loro valore assoluto.

Per quanto riguarda il numero di click si è registrato un calo medio del 29% mentre per quanto riguarda il numero di tasti premuti, il divario si assottiglia al 10%.

In conclusione, ci si può ritenere fondamentalmente soddisfatti dai risultati dei test per i seguenti motivi:

- I task che sono stati richiesti agli utenti si sono rivelati estremamente utili e pertinenti in quanto hanno permesso di far emergere un'importante criticità del plugin (criticità successivamente risolta ma non testata).
- I risultati ottenuti in termini di metriche e misurazioni si sono rivelati in ogni caso positivi: si hanno buoni motivi per credere che l'introduzione dell'uso di Light Studio per la configurazione dell'illuminazione delle scene in ambienti 3D possa effettivamente avere un impatto positivo su una produzione paragonabile a quella di Reverie Dawnfall.

5 Shading

Il presente capitolo costituisce il corpo centrale per quanto riguarda le diverse prove che sono state effettuate nell'ambito dello shading.

Si daranno per assodati i parametri e la nomenclatura caratteristici dei materiali, quali per esempio roughness, emission, diffuse, transparency, ecc.

Al fine di presentare dei risultati confrontabili, quando possibile, sarà utilizzato un modello 3D di riferimento, una sfera (detta shader ball) con due setup di luci che verranno mantenuti costanti attraverso tutte le prove:

1) Una luce direzionale bianca
2) Diverse area e point light colorate

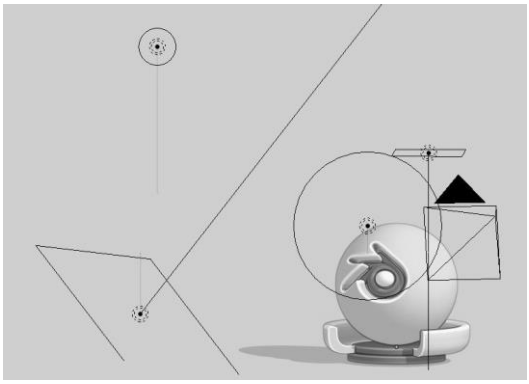


Figura 64: posizionamento delle luci colorate

Questo avverrà soprattutto per quanto riguarda i test eseguiti su Eevee, in quanto è una tecnica adottata successivamente. I test più vecchi sono stati invece eseguiti su scene di prova e modelli realizzati ad hoc. A causa dell'impossibilità di confronto fra dati raccolti quest'ultima modalità è stata abbandonata.

Nella prima sezione ci si concentrerà sulle prime prove di shading fatte in Cycles utilizzando le impostazioni nativamente disponibili. Nella seconda invece si passerà all'analisi di quanto fatto su Eevee. Nell'ultima sezione del capitolo verranno infine trattati i tentativi fatti, ma poi scartati, nell'ambito della generazione di pattern bidimensionali come griglie di esagoni e punti.

5.1 Test e tentativi passati

La ricerca dell'estetica per Reverie Dawnfall è iniziata nel 2018, da allora diversi sono stati i tentativi fatti, diverse le configurazioni provate, diverse quelle scartate. Durante questo processo però sono emerse ogni volta dei lati positivi che sono stati presi in considerazione per lo sviluppo della soluzione finale. Questo capitolo si passerà in rassegna le diverse prove eseguite nel corso del tempo fino al raggiungimento della soluzione attuale, soffermandosi sulle diverse scoperte fatte di volta in volta.

5.1.1 Cycles, screen space e compositing

Il primo approccio allo shading è stato fatto utilizzando il motore di rendering Cycles in quanto al momento delle prove non era ancora stato annunciato il lancio di Eevee.

Figura 65: scena di test

Essendo Cycles un motore di rendering raytracing (si veda il capitolo *Real time vs path tracing*), e a causa dell'impossibilità di agire sugli shader dopo la loro chiusura, sono poche le possibilità disponibili in fase di render. La ricerca di uno stile visivo alternativo era dunque limitato ai pochi strumenti disponibili.

I test di questo capitolo verranno eseguiti su una scena modellata appositamente con l'obiettivo di ottenere subito un riscontro visivo calato nell'ambito della serie.

Toon Shading

Per quanto riguarda i materiali disponibili, Cycles mette a disposizione nativamente un *Toon Shader*. Quest'ultimo è in grado di interpretare le informazioni luminose della scena dando automaticamente un risultato tipico da cartone animato: ombre nette, colore uniforme, riflessi definiti.

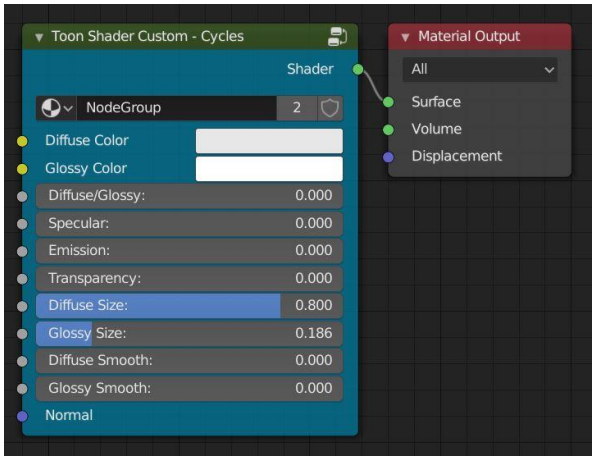


Figura 66: Toon Shader Cycles

Con l'idea di ottenere uno shader versatile e in grado di ottenere tutti i tipi di materiali possibili, si è quindi passati alla realizzazione di un setup di nodi in includere sia i comportamenti tipici di un materiale di tipo dielettrico che di un tipo metallico.

Nella figura a fianco si può notare il blocco funzionale finale con il quale l'artista può realizzare i materiali.

Si noti come i parametri esposti includano diversi parametri su cui agire.

- Diffuse Color: colore di base del materiale.
- Glossy Color: colore dei riflessi.
- Diffuse/Glossy: permette di scegliere quanto il materiale è lucido.
- Transparency: il grado di trasparenza del materiale.
- Emission: utilizzato quando il materiale emette luce.
- Specular: quanto il materiale è speculare.
- Diffuse Size, Glossy Size: definiscono la dimensione dell'area di ombra/riflessione.
- Diffuse Smooth, Glossy Smooth: specifica quanto il transiente fra luce e ombra è morbido.
- Normal: normali della superficie.

Nelle immagini seguenti sono riportate alcune varianti ottenibili modificando i parametri sopra descritti, in particolare quelli Diffuse/Specular e Transparency.

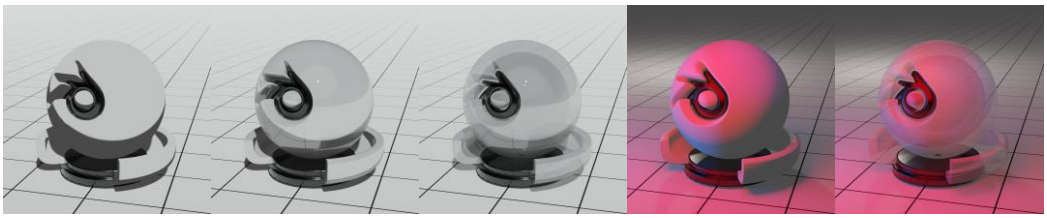


Figura 67: Da sinistra verso destra: diffuse, glossy, transparent; diffuse e transparent con luci colorate

Come si può notare dai risultati, il livello di stilizzazione diminuisce all'aumentare della quantità delle luci in scena, soprattutto quando queste differiscono in gamma cromatica. Vanno a perdersi infatti i transienti netti fra le zone di ombra e luce, nonostante i parametri di morbidezza dello shader siano settati al minimo.

Questo problema è in parte aggirabile scalando la dimensione delle fonti luminose, rendendole quasi puntiformi e forzandole così a gettare delle ombre nette. Ciò è però una soluzione parziale, in quanto si perde molta flessibilità in fase di illuminazione, inoltre Cycles produce immagini maggiormente rumorose al diminuire della dimensione delle luci.

Figura 68: risultato ottenuto utilizzando il Toon Shader

Rim light globale

Il passo successivo per aumentare la stilizzazione del risultato è stato quello di implementare un effetto rim light controllabile in maniera indipendente dalla luce.

Ispirandosi a quanto fatto in Gatta Cenerentola, si è deciso di utilizzare un approccio basato sul compositing in screen space.[16]
Ciò avviene sfruttando il passo di rendering *z-depth*, il quale associa ad ogni punto dell'immagine un valore pari alla distanza che questo ha dalla camera.

Una volta ottenuti questi dati è possibile duplicare la mappa e applicare un offset di qualche pixel lungo una direzione arbitraria. Successivamente basterà sottrarre fra di loro le due immagini, ottenendo così un effetto rim light che evidenzia i bordi degli oggetti in scena.

L'applicazione diretta di questa soluzione potrebbe non dare i risultati attesi a causa del fatto che i valori di z-depth non sono normalizzati. Durante i vari test si è notato che applicare una normalizzazione prima della sottrazione migliora il risultato estetico finale.

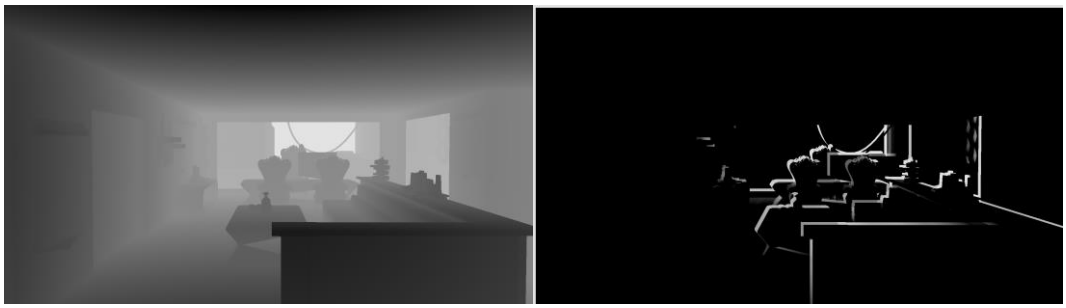


Figura 69: z-Depth a sinistra, rim light a destra

L'unico svantaggio dell'algoritmo è che durante l'operazione di offset e sottrazione viene a crearsi una zona di discontinuità sui bordi dell'immagine,

[16] https://youtu.be/fqPjmBRxqLI

risultando così in un bordo bianco. Questo perché spostando l'immagine duplicata viene a mancare una parte dell'informazione da sottrarre.
Ci sono due soluzioni possibili al problema:

- Eseguire un rendering a risoluzione superiore in modo da recuperare informazioni sui bordi (*overscan*).
- Mascherare la mappa della rim light sui bordi con una sorta di vignettatura, così facendo non vi sarà più la discontinuità, ma al contempo si perderà questa caratteristica sulle zone agli estremi dell'immagine.

Con lo scopo di evitare tempi di rendering maggiori, è stata scelta la seconda opzione.

Una volta ottenuta la mappa della rim light è possibile infine passare ad aggiungerla all'immagine con una semplice addizione, oppure può essere usata come maschera per aggiungere all'immagine degli effetti solo nelle zone dove essa è presente.

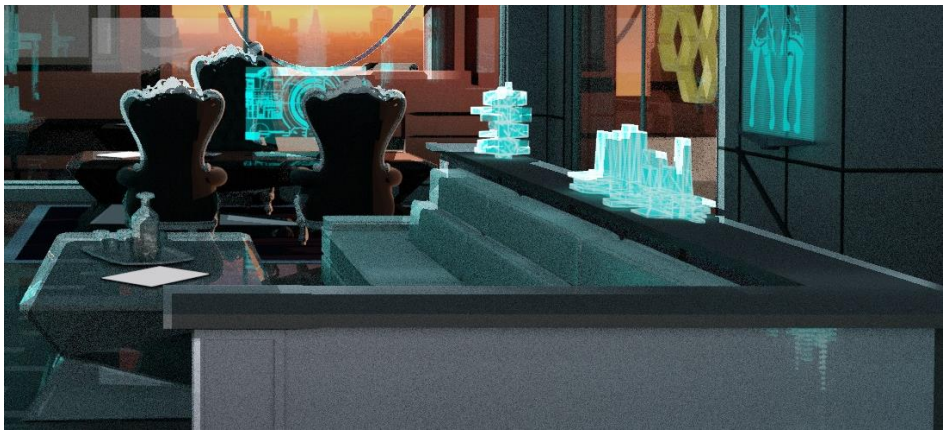


Figura 70: applicazione della rim light all'immagine

Freestyle

Essendo l'obiettivo quello di ricalcare il più possibile lo stile delle immagini di riferimento, il passaggio successivo è stato quello di tentare l'aggiunta di un tratto (*stroke*) di contorno agli oggetti in scena.
Fortunatamente Blender dispone di una funzione apposita che prende il nome di *Freestyle*.

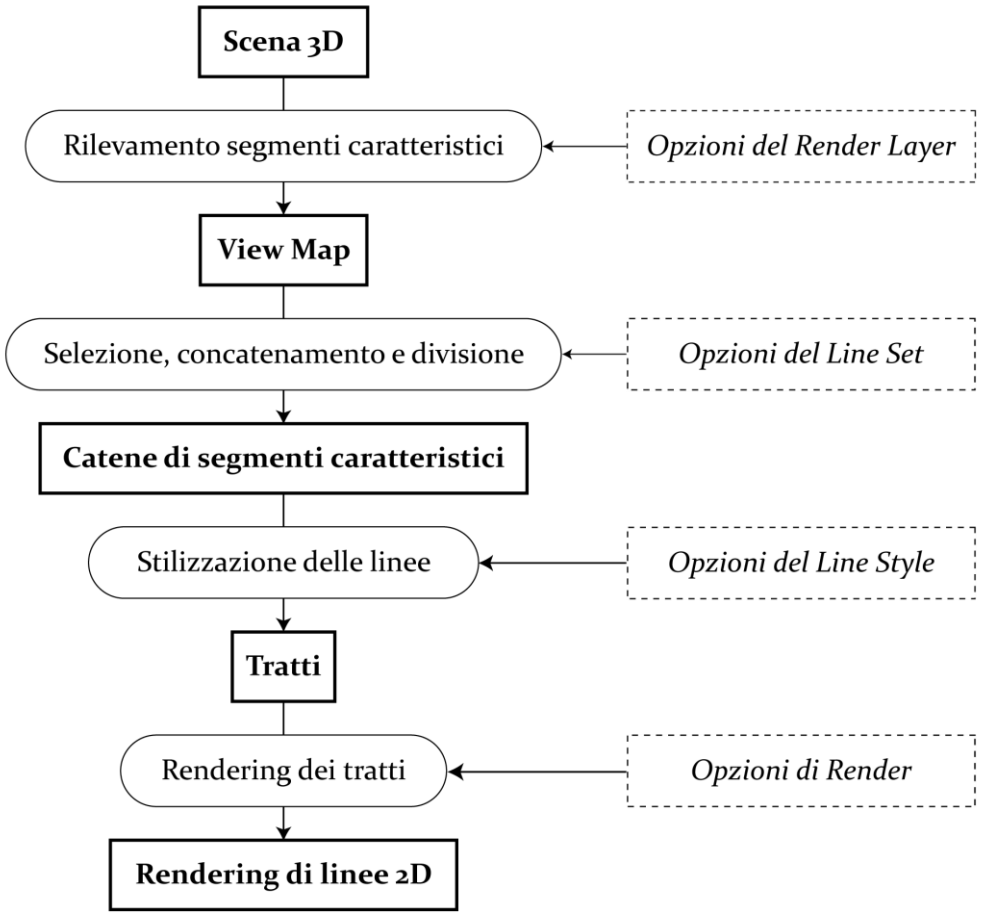


Figura 71: funzionamento del Freestyle

Il freestyle funziona definendo dei criteri di campionamento delle forme 3D in scena. A partire dai dati raccolti, l'algoritmo provvede alla generazione di linee alle quali vengono poi applicati degli effetti definiti dall'utente.[17]

La *Figura 71* illustra i principali passaggi dell'algoritmo che vengono qui di seguito spiegati.

- L'utente definisce i parametri dai quali verrà generata una mappa della vista di scena (*View Map*), questi comprendono la possibilità di considerare lo smooth shading durante i calcoli, l'angolo massimo fra due

[17] https://docs.blender.org/manual/en/latest/render/freestyle/index.html

facce perché queste vengano considerate nel calcolo, l'opzione di attivare il camera culling, le opzioni avanzate per l'algoritmo di calcolo dei *suggestive contours*.

- L'utente deve quindi specificare dei filtri tramite i quali verranno selezionate le linee sulle quali verranno disegnati i tratti. Fra questi vi sono:
    - Silhouette
    - Contorni esterni
    - Contorni suggestivi (*suggestive contours*)
    - Ridge e valley
    - Confini fra materiali
    - Spigoli marcati dall'utente
- Successivamente l'utente può decidere i parametri per modulare il tratto in ampiezza, lunghezza, precisione.
- A questo punto il motore di rendering si occupa di rasterizzare le linee calcolate.

L'immagine che segue mostra il ciò che si ottiene applicando l'algoritmo.

Figura 72: risultato della pass Freestyle

Per ottenere i risultati mostrati nell'immagine precedente si è scelto di utilizzare un filtro che considerasse solo i segmenti di silhouette e le pieghe superiori ai 90°. Si è inoltre applicato un effetto calligrafico al tratto, in modo che lo spessore del tratto diminuisse con la lunghezza dello stesso. Così facendo si è ottenuta un'estetica pulita e definita, non particolarmente fedele alle immagini di riferimento, le quali invece presentano un tratto più simile ad uno schizzo. Nel prossimo paragrafo si affronterà il metodo con cui si è spezzata questa regolarità.

Per concludere va sottolineato come l'utilizzo di questa tecnica rallenti di parecchio la fase di rendering, questo soprattutto per le scene più complesse, fino ad arrivare a situazioni in cui il programma va in stato di stallo.
Verranno considerate nel futuro opzioni più veloci, sebbene ancora sperimentali.

Edge Displacement

Uno degli elementi caratterizzanti in una produzione 3D è l'entità discreta dei modelli stessi. Il fatto che i bordi degli oggetti siano spigolosi e perfettamente definiti può essere positivo quando si tenta di modellizzare un fenomeno reale, al contrario quando si tenta di stilizzare questo diventa un enorme fattore limitante.

Sempre grazie all'esempio portato da Gatta Cenerentola, si è pensato di implementare un algoritmo di edge displacement: applicando una distorsione ai bordi dell'immagine finale è possibile in parte mitigare l'effetto sopra descritto.

Il primo passaggio è quello di definire un modo per rilevare i bordi delle forme dell'immagine. Ciò si può realizzare grazie ad un classico algoritmo di *edge detection*, la versione disponibile in Blender è quella di un filtro Laplaciano.

Il punto successivo è dunque quello di definire il set di dati dai quale partire per trovare i bordi ai quali si è interessati. Si potrebbe pensare di sfruttare il

risultato finale del rendering (beauty pass) e applicare così il filtro su di essa. Il problema che sorgerebbe sarebbe però quello di portare in risalto delle aree di contrasto ad alta frequenza dovute anche a texture e illuminazione, provocando così un displacement anche in zone interne agli oggetti di scena, e non solo ai loro bordi.

Dopo diversi test, si è deciso di sfruttare una somma delle informazioni incluse nella z-depth e nelle normali dei modelli. Così facendo, con una è stato possibile ottenere le informazioni di profondità e distanza, con l'altra le differenze di inclinazione fra una faccia e l'altra delle geometrie in scena.
Applicando il filtro Laplaciano sulla mappa così ottenuta porta all'individuazione dei bordi ricercati.

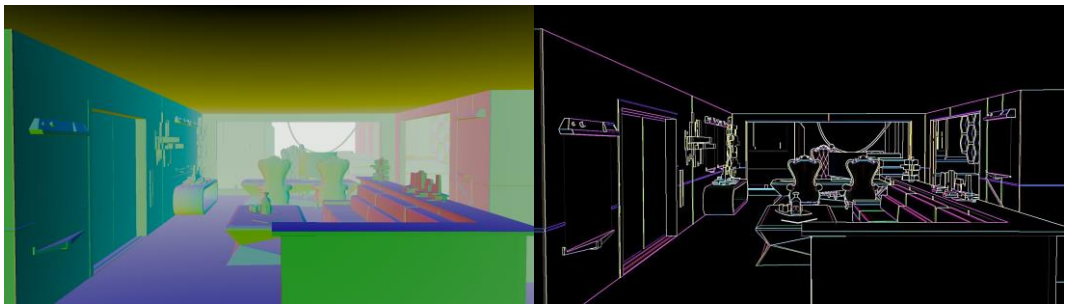


Figura 73: somma di z-depth (normalizzata) e normali a sinistra, risultato del filtraggio a destra.

Stabilita la maschera è quindi possibile applicare un'operazione di displacement nella zona designata applicando un offset ai singoli pixel secondo i valori generati da un Perlin Noise. Nel caso in cui si voglia aumentare l'area d'influenza della distorsione basta applicare un filtro Gaussiano alla maschera.

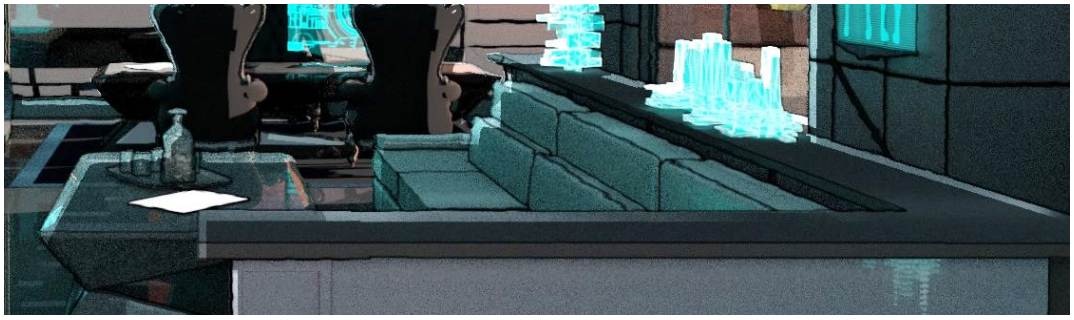


Figura 74: risultato del displacement applicato ai bordi

Il procedimento può essere applicato all'immagine finale o anche solo ai tratti generati con il Freestyle. Il risultato si avvicina molto di più alle immagini di riferimento e ha una resa molto buona soprattutto sulle immagini statiche.

Per quanto riguarda le immagini in movimento invece, l'utilizzo di un noise statico sovraimposto ad un'immagine in movimento può talvolta risultare fastidioso all'occhio: quando l'immagine si muove rapidamente si rischia una sensazione di strobing, quando i movimenti sono lenti il risultato è simile ad un riflesso in uno specchio d'acqua.
Un modo per attenuare l'effetto è quello di animare a passo due il noise, questo però ha forti ripercussioni sullo stile di animazione.

Setup finale di compositing

Tutti i livelli ottenuti possono quindi essere combinati in compositing.

A quanto già calcolato in precedenza si aggiungono dei passi di render di ambient occlusion, riflessioni e ombre. Su quest'ultime in particolare si sono fatte diverse prove tentando di abbinare dei pattern alle ombre. Ciò non ha avuti riscontri positivi e l'idea è stata quindi scartata momentaneamente, per poi essere ripresa successivamente con tecniche differenti.

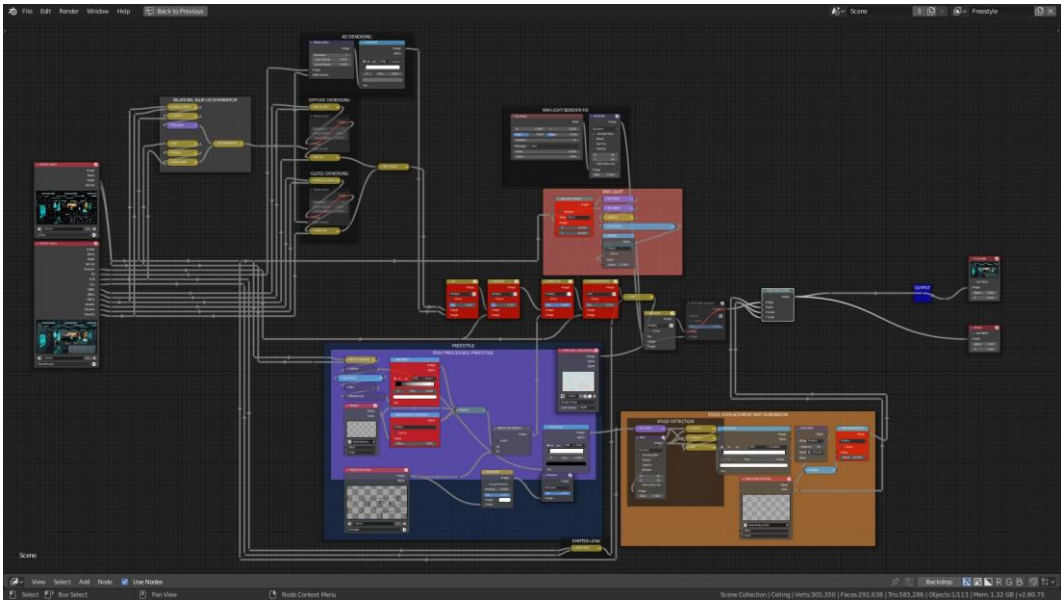


Figura 75: setup di compositing a nodi

Osservazioni

Sebbene i risultati ottenuti fossero discretamente buoni, la soluzione è stata scartata a causa della necessità di lavoro in post-produzione che al momento dei test è stato giudicato eccessivo. L'obiettivo era quello di avere un risultato immediato senza la necessità di ulteriore intervento manuale. Il dover impostare delle soglie per i diversi filtri a seconda della situazione, l'essere costretti ad eseguire la maggior parte delle operazioni internamente al compositor di Blender (con conseguenti restrizioni nella flessibilità del workflow dello studio) e gli elevati tempi di rendering hanno fatto sì che venissero considerate altre opzioni.

Nel frattempo, l'annuncio del nuovo motore di rendering Eevee ha fatto sì che le ricerche si spostassero sul branch di Blender 2.8.

Durante le ricerche effettuate su Cycles si sono quindi esaminate le possibilità offerte dal compositing e dalle innumerevoli pass di rendering che il motore è in grado di offrire. Elementi come il Freestyle, l'edge displacement e la mascheratura dei bordi saranno comunque considerati nelle future iterazioni di ricerca.

Figura 76: risultato finale

5.1.2 Eevee, object space e Shader to RGB

A inizio 2018 si decide dunque di muovere i nuovi test su Eevee, in quel momento disponibile solo in versione Alpha. L'idea di poter ottenere feedback visivi immediati anche per elementi come capelli, volumi e particellari ha subito raccolto l'interesse della produzione.

Sfortunatamente, in Eevee non è inizialmente presente un Toon Shader o un suo equivalente, le uniche possibilità di ottenere un'estetica stilizzata erano ridotte all'utilizzo di calcoli basati sulle posizioni delle luci, trascurando di fatto ogni parametro dovuto alla luminosità e colore delle stesse.
Fortunatamente il 16 maggio 2018 viene introdotto in Blender il nodo *Shader to RGB* il quale porterà con sé un'infinità di nuove possibilità da esplorare.[18] I paragrafi seguenti si concentreranno quindi sui test eseguiti esplorando le potenzialità offerte da Eevee.

Shader to RGB

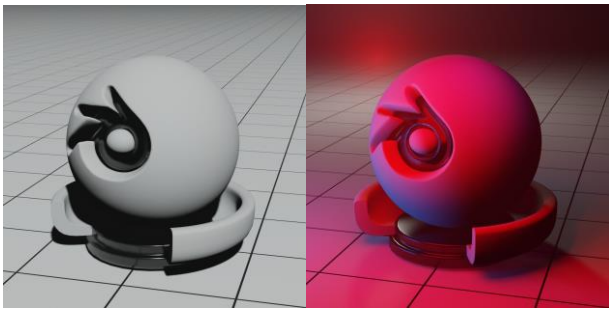


Figura 77: uno shader diffuse standard illuminato con luce direzionale e luci colorate di tipo area

Il nodo in questione introduce una vera e propria rivoluzione nell'ambito di shading di Blender. Brevemente, con questo nodo è possibile ottenere i dati successivamente alla chiusura di un nodo di shading, potendo eseguire così operazioni sui colori calcolati dal modello di illuminazione.[19]

Nella *Figura 77* è riportato il risultato ottenibile in Eevee utilizzando un semplice shader diffuse con colore bianco illuminato con una singola luce direzionale e diverse luci area colorate.

---

[18] https://youtu.be/l3VGCUhN4BU
[19] docs.blender.org/manual/en/latest/render/shader_nodes/converter/shader_to_rgb.html

Collegando il nodo Shader to RGB all'output è quindi possibile trattare i dati come un normale stream di tipo RGB.

Per esempio, si può provare a collegare una Color Ramp eseguendo così un'operazione di soglia sui valori di luminosità.

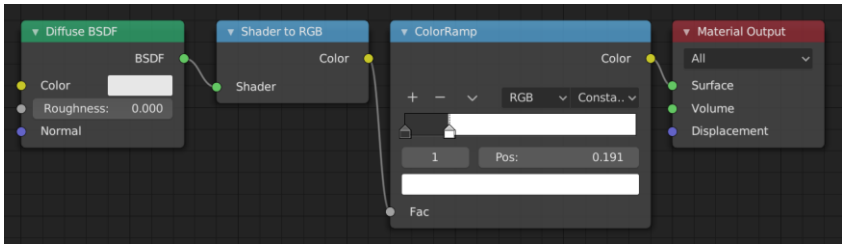


Figura 78: operazione di soglia (clamping)

Il risultato così ottenuto si avvicina molto allo shader di tipo Toon che si ha a disposizione in Cycles ma con un grande compromesso: come si può osservare dalle immagini seguenti, non sono preservate le informazioni riguardanti il colore delle luci.

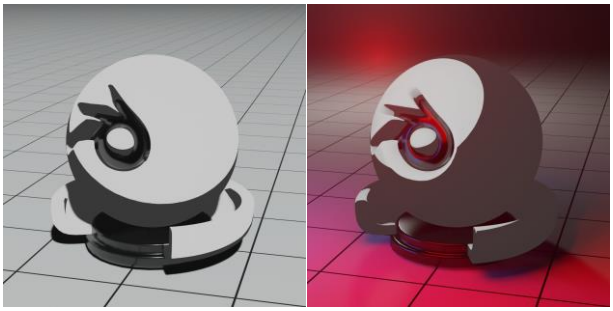


Figura 79: risultato dell'operazione nelle due situazioni di luce

Questa è solo il primo passo verso gli esperimenti successivi, ma continuerà a costituire le fondamenta per tutti i test.

Mascheramento di texture in object space

L'idea successiva è stata quella di utilizzare i valori in scala di grigio così ottenuti come maschera per rivelare delle texture presenti sul modello.

Dopo diversi test durante i quali gli alberi di nodi sono diventati via via più complessi, si è giunti ad una prima forma di sintesi.

I requisiti da soddisfare al fine di realizzare delle prove approfondite su modelli più complessi erano i seguenti:

- Modularità.
- Possibilità di applicare le texture in modo automatico, senza distorsione e agendo il meno possibile sui singoli modelli.
- Propagare a tutti i modelli le modifiche effettuate alle proiezioni e alla scala delle texture in modo automatico.

Si è quindi deciso di utilizzare due blocchi funzionali separati, uno per le ombre e uno per i punti di luce. I blocchi dispongono di parametri ed impostazioni interne che sono condivise tra tutte le istanze.

Volendo emulare lo stile dei disegni di riferimento si è scelto di fare dei tentativi utilizzando del noise di tipo Perlin unito ad un Voronoi di scala molto ridotta per le ombre, e una texture costituita da un'immagine con pennellate per le luci.
Le texture sono generate a partire da coordinate Object sfruttando una proiezione di tipo Box per la texture con le pennellate. In questo modo l'immagine è proiettata lungo i tre assi coordinati (proiezione triplanare), compensando l'assenza di UV e avvolgendo omogeneamente il modello senza distorsioni.

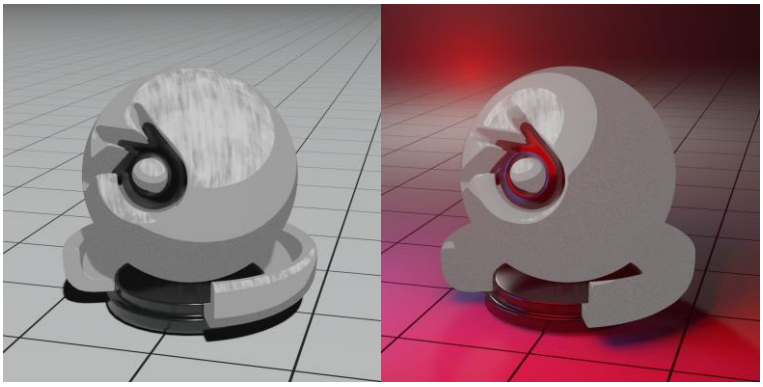


Figura 80: texture masking applicato ad uno shader emissivo

Il risultato è stato poi collegato direttamente ad uno shader emissivo. Come si può notare dall'immagine precedente però, il contributo del colore delle

luci è completamente trascurato. Si può dunque tentare di ovviare il problema collegando il risultato ottenuto, come se fosse una texture, ad un Principled Shader, unendo così il recupero del contributo delle luci alla possibilità di avere la *texture* di base di un modello *che varia con la luce*. Quest'ultima è una cosa tutt'altro che banale, precedentemente impossibile in Cycles e tuttora impossibile nella gran parte dei motori di rendering.

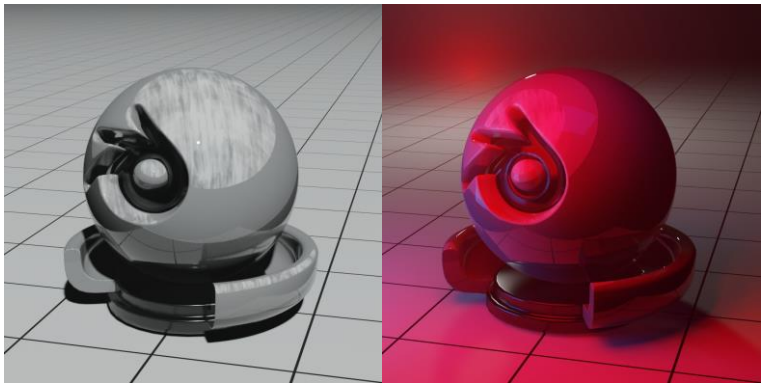


Figura 81: texture masking applicato ad uno shader emissivo

Il risultato è sicuramente interessante ma presenta dei problemi:

- Le texture così mappate appaiono statiche sul modello quando questo viene animato, il risultato è molto simile ad un videogioco 3D in Cel Shading, come per esempio *Borderlands*.[20] Questo non è ottimale in quanto il risultato che si vuole ottenere è più organico e simile ad un fumetto animato.
- Il controllo dello shader è complesso in quanto vanno gestiti separatamente i parametri non fotorealistici e fotorealistici, con conseguenti risposte differenti a seconda delle luci di scena.

Per tentare di affrontare il primo punto si è provato ad usare una proiezione in screen space. La situazione purtroppo non migliora, anzi, la permanenza sullo schermo di un pattern così caratteristico e unico mentre i soggetti si muovono provoca una situazione di fastidio.

[20] https://borderlands.com/en-US/

Si è infine tentato di animare il pattern di pennellate, ma il risultato è difficilmente gestibile, portando dei risultati interessanti che potrebbero essere sostenuti in un corto d'animazione, ma non per un prodotto seriale.

Rim light in object space

Il passaggio successivo è quello di simulare una rim light in object space, riducendo il più possibile gli step di compositing.

Si sono trovati due modi per affrontare il problema.

Normal input

È possibile manipolare le normali attraverso il nodo Normal, questo dispone di una sfera ruotabile dall'utente. Agendo su di essa è possibile orientare un vettore. Dall'output Dot uscirà quindi il prodotto scalare fra le normali in ingresso e il vettore manipolato dall'utente.

Ruotando opportunamente la sfera si potrà ricreare l'effetto di una luce proveniente da dietro il soggetto. Per aumentare la stilizzazione si può aumentare il contrasto dell'effetto ottenuto usando la solita Color Ramp per eseguire un clamping dei valori.

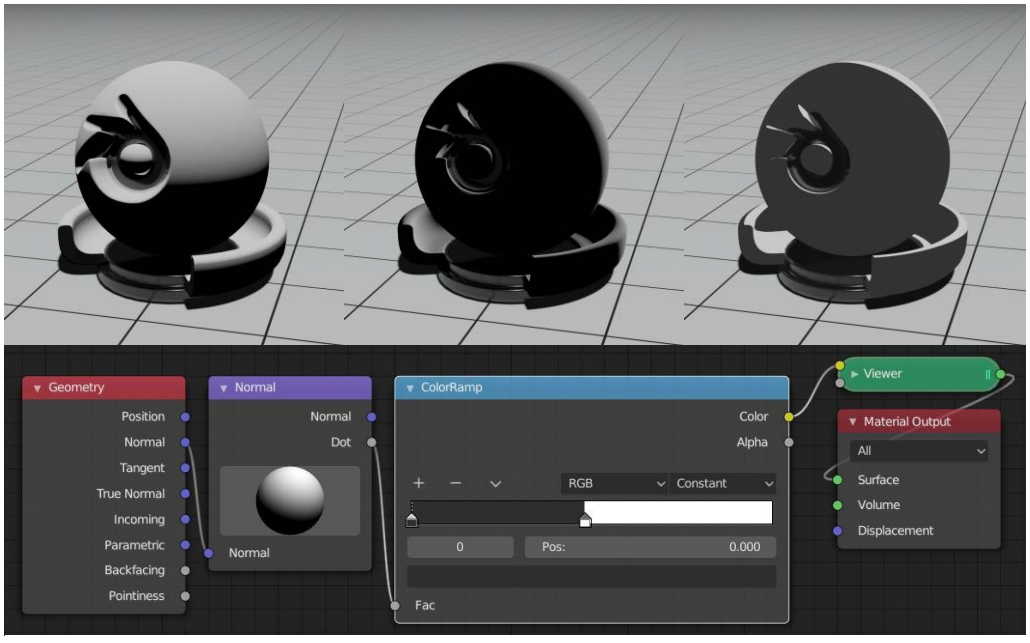


Figura 82: illustrazione dei diversi passaggi con i relativi nodi

Fresnel input

La seconda opzione prevede l'utilizzo dei valori di Fresnel. Questi valori sono tanto maggiori quanto l'angolo fra il vettore di vista e quello normale alla superficie è maggiore. Si può quindi effettuare un'operazione di clamping ottenendo così un effetto simile ad una rim light.

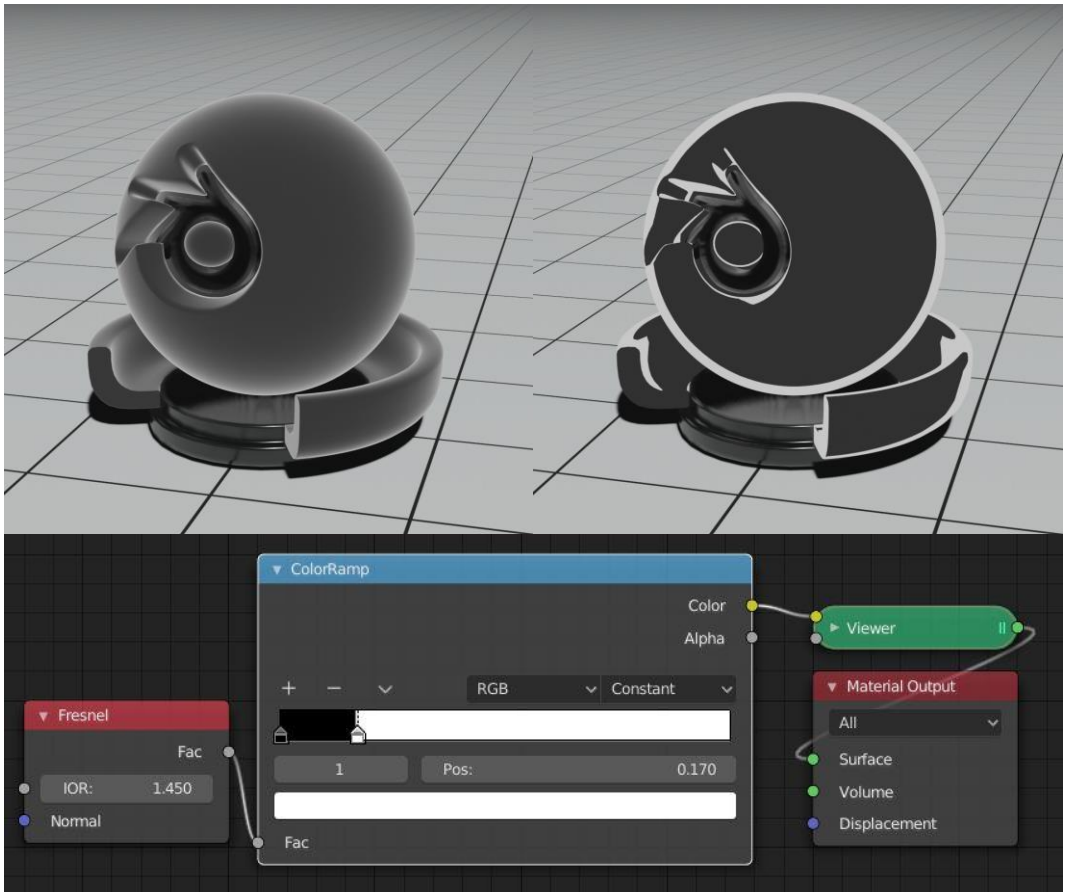


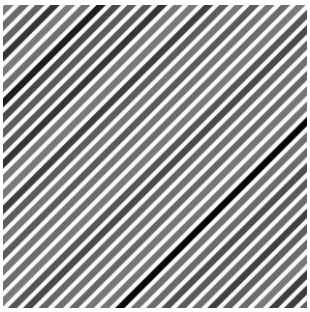
Figura 83: i due passaggi

Come si può notare dall'immagine però la luce non ha direzione e avvolge in modo uniforme il modello. Questo comportamento può essere modificato mascherando i valor di Fresnel con quelli delle normali ottenuti al paragrafo precedente. Oppure si possono usare dei valori di luminosità ottenuti con un nodo Shader to RGB. La seconda opzione verrà preferita per il setup finale in quanto più pulita e regolare, anche se va evidenziato come su superfici piane di grandi dimensioni possano emergere problemi dovuti all'assenza di curvatura.

5.1.3 Pattern 2D

Con i risultati ottenuti ai passaggi precedenti, unita all'ispirazione avuta da *Spiderman – Into the Spiderverse*, è iniziata ad emergere la consapevolezza che l'utilizzo di pattern simili ad un fumetto generati in screenspace sarebbero potuto essere un elemento interessante da esplorare.

Retino a righe



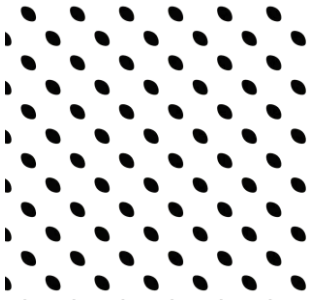
Si è iniziato dall'elemento più semplice possibile, ovvero un retino costituito da linee parallele a spessore variabile. L'implementazione è abbastanza diretta, come già visto Blender dispone nativamente di una texture di tipo Brick. Manipolandone opportunamente i parametri si riesce ad ottenere subito l'estetica ricercata. In particolare, basta semplicemente ridurre l'altezza dei mattoni e utilizzare quindi il valore del mortar per stabilire la dimensione dei tratti.

Tramite l'uso di un nodo mapping è successivamente possibile orientare il pattern, in questo caso è stata applicata una rotazione di 45°.

Retino puntinato

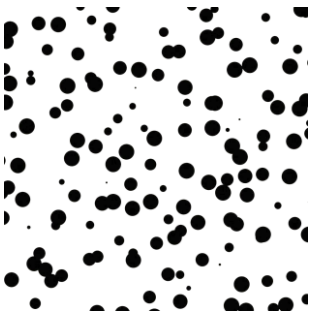
Il secondo pattern da realizzare consiste in una griglia di punti. Vi sono diversi metodi per raggiungere lo stesso risultato, alcuni più semplici, altri più complessi.

Magic Texture



Con questa modalità è possibile ottenere un pattern sfruttando la texture Magic (Depth: 0, Distortion: 1) effettuando un'operazione di clamping in modo da ottenere una griglia di punti. Si notano però subito due problemi. Gli elementi non sono effettivamente

circolari, al contrario sono più simili a degli ovali. Inoltre, non si ha controllo sulla scala dei singoli punti a causa della natura del noise di partenza: modificando l'operazione di clamping si ottengono delle strisce ondulate invece che dei puntini più grandi. Questa modalità viene quindi subito scartata.



Una versione simile prevede l'utilizzo di una texture di tipo Voronoi, ma i problemi che si presentano sono simili, inoltre i punti così ottenuti hanno un posizionamento casuale e non ordinato. Volendo ottenere un risultato regolare, anche questa idea viene scartata, nonostante la sua semplicità.

Griglia regolare di coordinate UV

Un secondo approccio, più avanzato e controllabile, prevede la manipolazione delle coordinate screen space realizzando una ripetizione di queste. A partire dalla griglia di coordinate così ottenuta si può ottenere una ripetizione di una qualsiasi texture desiderata, in quanto i nodi di texture procedurali a parità di valori d'ingresso restituiranno sempre gli stessi risultati in uscita.

Si consideri il caso monodimensionale. Le coordinate si estendono lungo l'asse delle x per valori che vanno da 0 a 1 (come al solito, qui rappresentati come colori). Moltiplicando per 5 si otterrà che i valori saranno mappati da 0 a 5. Applicando quindi un'operazione *Fract* (definita come `x - floor(x)`) si otterrà cinque volte una ripetizione dei valori da 0 a 1.

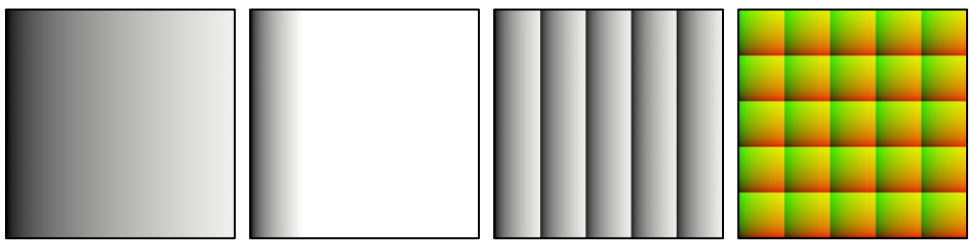
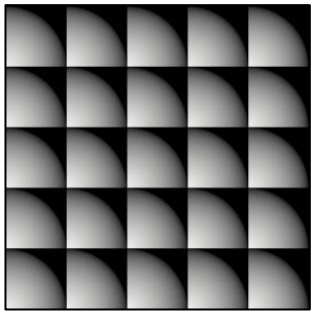


Figura 84: il risultato dei diversi passaggi e infine l'unione dello stesso procedimento applicato per l'asse y, si noti come la seconda immagine sia totalmente bianca a causa della non rappresentabilità dei valori

Il processo può essere ripetuto anche per l'asse y, combinando poi i valori.

Le coordinate così ottenute possono essere utilizzate per generare pattern ripetuti, a patto che questi siano definiti nell'intervallo (0, 1).



Si può dunque provare a collegare quanto fatto finora ad una texture *Gradient* di tipo sferico. Si otterrà così quanto riportato nella figura a sinistra. Questo perché la funzione Gradient sferico restituisce una sfera di raggio 2 centrata nell'origine, quindi con estensione che va da -1 a 1. Basta quindi moltiplicare le coordinate sopra ottenute per 2 e sottrarre 1, in modo da ottenere delle coordinate che in ogni cella vanno da -1 a 1.

Al pattern così ottenuto può essere applicata una soglia in modo da poter gestire la dimensione dei punti. Il risultato è riportato nella figura.

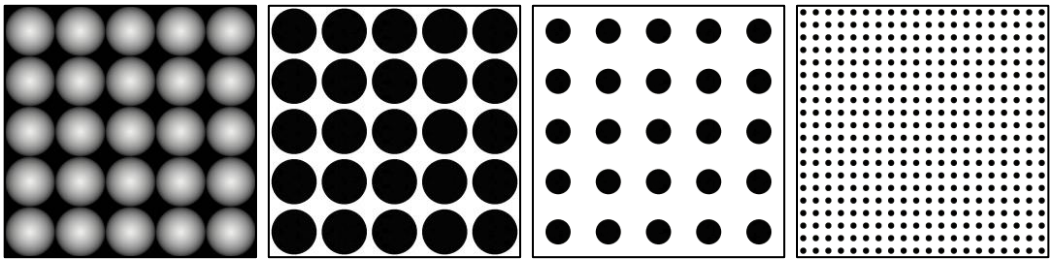


Figura 85: i gradienti sferici e alcune varianti delle versioni con clamping, sia per scala dei punti che per scala del pattern a livello globale

Riassumendo, dato il fattore di scala α, lo spazio trasformato S'_{xy} si ottiene con la seguente formula:

$$S'_{xy} = 2fract(\alpha S^0_{xy}) - 1$$

La soluzione è semplice ed efficace, però la griglia è formata da quadrati. Ai fini di realizzare un classico retino da fumetto i puntini dovrebbero essere disposti su una griglia costituita da rombi. Questo porta delle complicazioni notevoli in quanto i quadrati generanti le coordinate sono spaziati in modo irregolare e presentano possibili sovrapposizioni. Si sono comunque fatti dei tentativi in questa direzione.

Unione di griglie rettangolari con offset

Quanto illustrato al passo precedente può essere ripetuto generando due griglie rettangolari (invece che quadrate) e applicando un offset opportunamente scelto ad una delle due. Unendo i punti così ottenuti si può ottenere una disposizione su griglia triangolare.

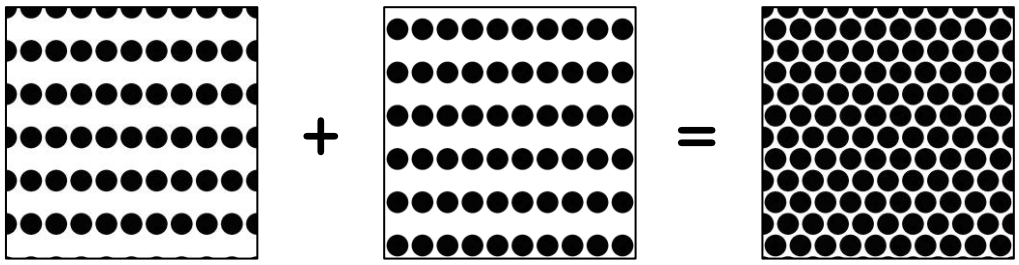


Figura 86: unione delle due griglie rettangolari

Al fine di alterare il rapporto fra coordinate orizzontali e verticali, invece che utilizzare Fract, è possibile sfruttare la funzione *Modulo*: dati gli input x e y, l'operazione `mod(x, y)` restituisce il resto ottenuto dividendo x per y. Per generalizzare, si può dire che Fract è un caso particolare di Modulo:

```
fract(x) = mod(x, 1)
```

Date le due griglie S^0_{xy} e Q^0_{xy} di partenza e il fattore di scala α le operazioni diventano quindi:

$$\begin{pmatrix} S'_x \\ S'_y \end{pmatrix} = \begin{pmatrix} 2mod_{1.0}(\alpha S^0_x) - 1 \\ 2mod_{1.7}(\alpha S^0_y) - 1 \end{pmatrix}$$

$$\begin{pmatrix} Q'_x \\ Q'_y \end{pmatrix} = \begin{pmatrix} 2mod_{1.0}(\alpha(Q^0_x + 0.5)) - 1 \\ 2mod_{1.7}\left(\alpha(Q^0_y + \frac{\sqrt{3}}{2})\right) - 1 \end{pmatrix}$$

Le costanti additive per l'offset (0.5, $\frac{\sqrt{3}}{2}$) sono calcolate appositamente in modo da ottenere una griglia di triangoli equilateri.

La *Figura 87* riporta l'implementazione in Blender.

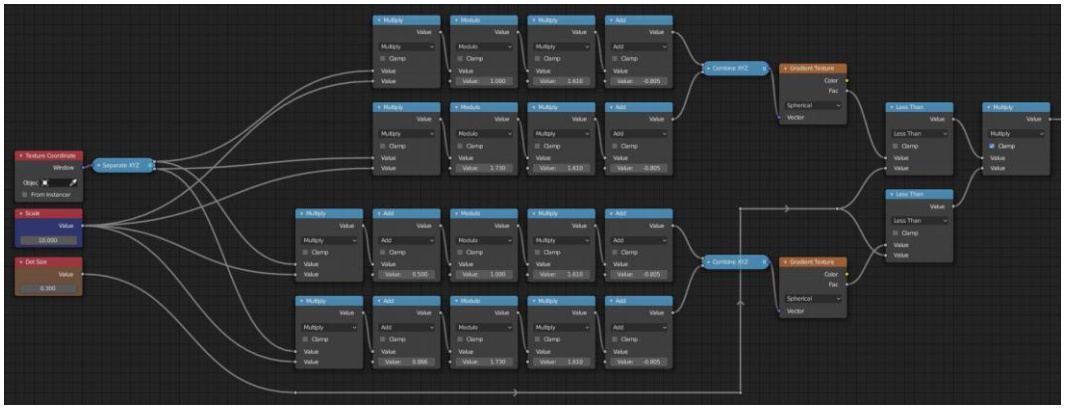


Figura 87: l'implementazione su Blender

Griglia unica con offset e generazione tramite distance field

Vi è infine un ultimo metodo che è stato considerato per la realizzazione del retino. Se si vuole trovare una correlazione con una delle modalità precedenti, quella che più si avvicina concettualmente è la versione che sfrutta il pattern Voronoi. Come già si è detto però, alla corrente implementazione di Blender, non è possibile gestire la disposizione dei punti da cui il campo di distanze (*distance field*) è generato. Si può tentare quindi di realizzare una versione più semplice e controllata, un caso specifico più facile e in grado di restituire dei risultati utili.

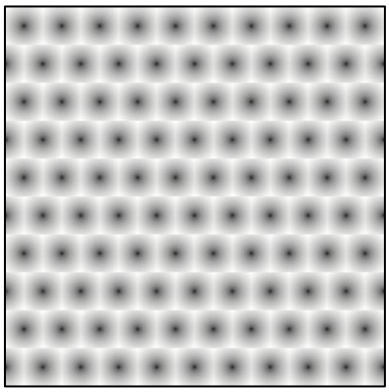


Figura 88: distance field desiderato

L'obiettivo è quindi quello di ottenere delle caselle quadrate disposte con un pattern simile a quello di un muro a mattoni regolarmente alternati. Considerando poi il centro di ogni casella basta calcolare la distanza fra esso e ogni altro punto della casella stessa, ottenendo così un gradiente circolare al quale si può successivamente applicare la solita operazione di clamping al fine di avere un punto di raggio a scelta.

Questo a livello intuitivo può essere molto semplice da immaginare, parlando però di fragment shader, e non avendo altro da cui partire se non le sole coordinate Screen Space, non sarà possibile applicare direttamente quanto detto.

Si può iniziare a pensare come ottenere un offset di riga. Volendo "spostare" un'intera sezione di spazio, ciò può essere realizzato sommando delle sezioni di piano a valore costante. Si realizza dunque un pattern a strisce orizzontali alterne: una metà a valore nullo, l'altra metà con un valore pari all'offset desiderato. Ciò si può ottenere in diversi modi, impiegato nella soluzione prevede una partizione verticale dello spazio con il metodo Fract, successivamente applicando un'operazione *Less Than*[21] si può ottenere l'alternarsi di righe bianche e nere a valori 0 e 1 rispettivamente. Moltiplicando il tutto per una costante si può così stabilire l'offset che andrà sommato alla componente x delle coordinate.

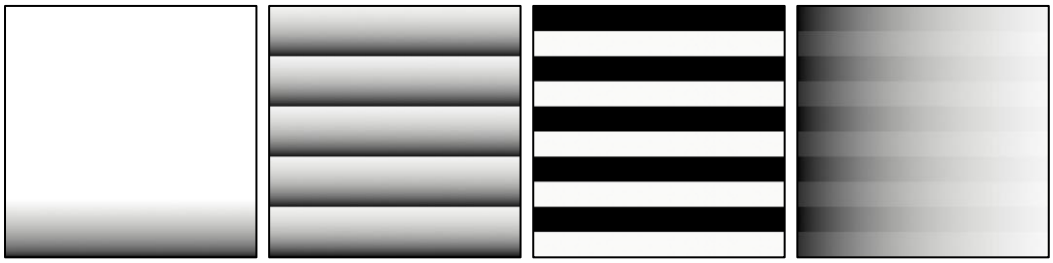


Figura 89: generazione dell'offset e somma alle coordinate x

Definendo δ_x l'offset orizzontale da applicare, α il fattore di scala verticale e O_x l'offset spaziale da applicare alle coordinate, quanto descritto può essere condensato in:

$$O_x = \begin{cases} \delta_x, & \alpha_y fract(S_y^0) < 0.5 \\ 0, & \alpha_y fract(S_y^0) \geq 0.5 \end{cases} \text{ con } \alpha_y = \frac{1}{2\alpha}$$

Giunti a questo punto, è possibile effettuare un partizionamento delle coordinate x utilizzando Fract nuovamente. Si calcolano anche i valori per y.

Per α valore di scalamento si ha:

$$S'_x = fract(\alpha(S_x^0 + O_x))$$

$$S'_y = fract(\alpha(S_y^0))$$

[21] L'operazione `lessThan(x,y)` restituisce 1 quando il valore di x è minore di y, 0 viceversa.

Così facendo si è ottenuta una griglia di coordinate con offset a righe alternate. Il risultato si può osservare nella *Figura 90*.

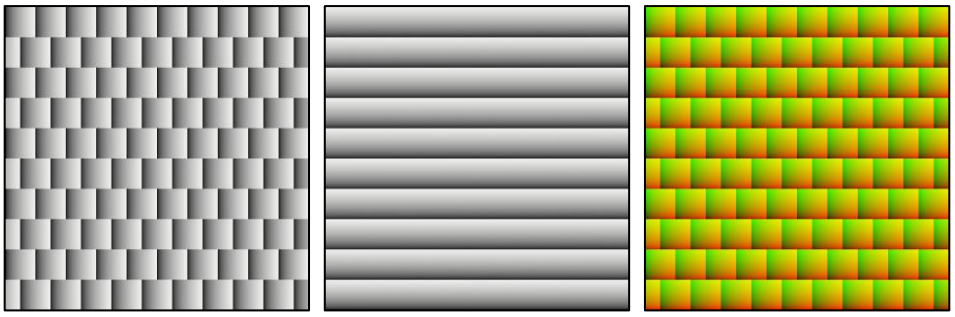


Figura 90: coordinate x, y e la loro unione

Visto che le coordinate per ogni cella vanno da 0 a 1, non resta che calcolare la distanza fra il punto medio di coordinate (0.5, 0.5) e il resto dei punti appartenenti alla cella, ottenendo così un distance field.

Riassumendo, il pattern finale è ottenibile applicando una soglia a P, dove quest'ultimo è ottenibile con il metodo:

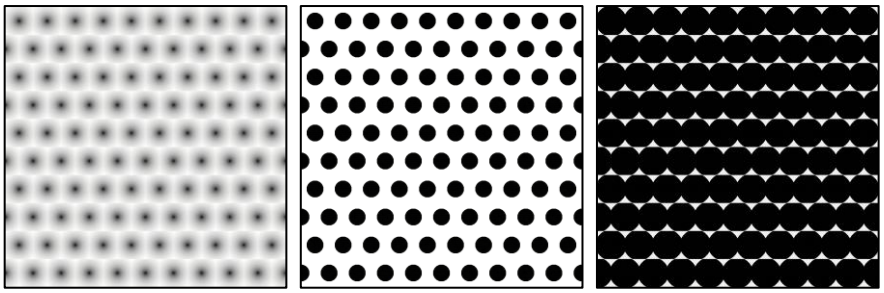
$$P = \sqrt{(S'_x - 0.5)^2 + (S'_x - 0.5)^2}$$

$$S'_x = fract(\alpha(S_x^0 + O_x))$$

$$S'_y = fract(\alpha(S_y^0))$$

$$O_x = \begin{cases} \delta_x, & \alpha_y fract(S_y^0) < 0.5 \\ 0, & \alpha_y fract(S_y^0) \geq 0.5 \end{cases} con\ \alpha_y = \frac{1}{2\alpha}$$

Viene riportato nelle immagini seguenti il risultato così ottenuto. Si noti come vi siano dei problemi quando la dimensione dei punti necessiterebbe una sovrapposizione fra le diverse righe, ciò però non può avvenire a causa della limitatezza delle celle utilizzate.



La soluzione esposta, nonostante le limitazioni sui punti di dimensione maggiore, è la più leggera da calcolare poiché non necessita dell'utilizzo di nodi di tipo texture, più pesanti computazionalmente.

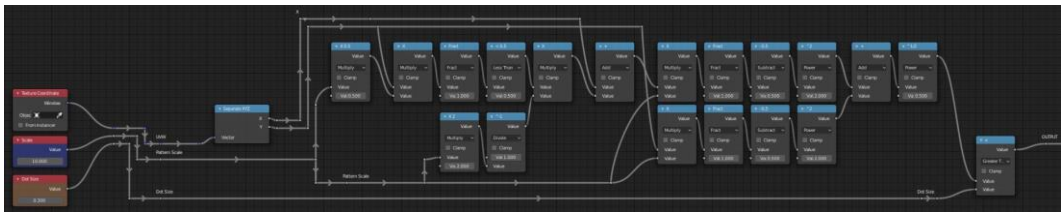


Figura 91: implementazione in Blender

Griglia esagonale

Essendoci in Reverie Dawnfall diversi riferimenti al mondo degli insetti, si è pensato che fosse interessante utilizzare alcuni pattern che lo ricordassero. Si è dunque deciso di provare a realizzare un pattern ad alveare. I requisiti per la realizzazione avrebbero dovuto prevedere che:

- il pattern fosse scalabile in dimensione;
- ogni singola cella potesse essere scalata indipendentemente;
- ad ogni cella venisse assegnato un colore casuale.

Come al solito, si è prima fatta una ricerca per vedere le soluzioni già presenti in letteratura. Si sono trovati due approcci: uno prevede la generazione di un esagono in coordinate polari[22] con conseguente ripetizione su griglia, il secondo utilizza un metodo che si rifà al già citato Voronoi.
A causa dell'eccessiva complessità e della difficoltà ad implementare un colore casuale, l'opzione che sfrutta le coordinate è stata scartata dopo qualche prova. Si è deciso di usare un'altra volta un approccio basato su distance field, sfruttando però il più possibile le simmetrie che l'esagono offre.

[22] https://thebookofshaders.com/07/

Simmetria ed elemento di base

Esaminando un pattern a nido d'alveare si nota subito come si possano evidenziare subito alcune simmetrie. Nel caso dell'implementazione che si è infine scelta di realizzare, la sezione più piccola che si è voluta considerare è quella riquadrata di blu nella *Figura 92*. Si noti come si potrebbero considerare aree più piccole, ma serve fare un compromesso fra la complessità della realizzazione dell'elemento di origine e quella che sarà la griglia di coordinate da generare al fine di poter specchiare e replicare l'elemento stesso.

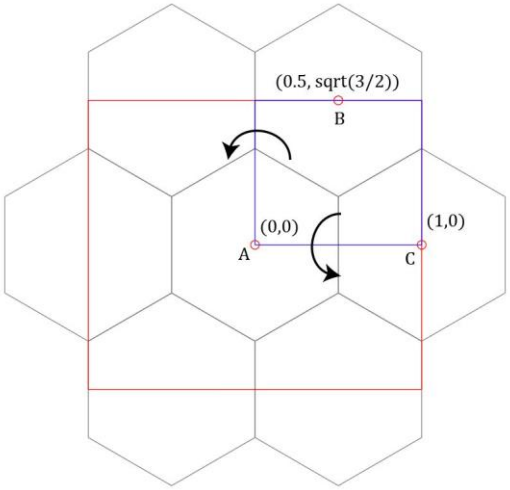


Figura 92: pattern esagonale e le sue simmetrie

Si osservi come ciò che è contenuto nel quadrato blu possa essere ribaltato sull'asse orizzontale per ottenere la continuazione del pattern in verticale, per ottenere una continuazione in orizzontale è sufficiente invece copiare il contenuto, senza necessità di specchiare.

Scelta dunque l'unità minima del pattern non resta che generarla e trasformare le coordinate al fine di ottenere la replicazione desiderata. Va sottolineato che questa operazione, sebbene faciliti di parecchio la generazione del pattern di base, renderà abbastanza complessa la gestione dei colori delle singole celle, poiché si rivelerà necessario trovare un modo per assegnare un indice ad ogni cella, nonostante queste vengano di fatto copiate in modo indistinto.

Si consideri dunque il contenuto del quadrato blu. Questo può essere ottenuto con la generazione di tre celle di un pattern Voronoi con i punti sorgente posizionati alle coordinate evidenziate nella *Figura 92*. Va quindi implementato un codice che permetta di generare i distance field relativi ai tre punti,

di confrontarli e di scegliere di conseguenza le relative aree di influenza, restringendole alle sezioni di piano ove la distanza dal punto in questione è minima.

Per semplicità espositiva, i passaggi saranno illustrati con figure. Si considerino i punti A, B e C, disposti come in *Figura 92*, e il loro distance field ottenuto come prodotto scalare delle coordinate Screen Space traslate nel punto con se stesse.

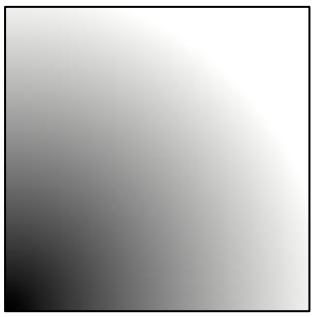


Figura 93: distance field di A

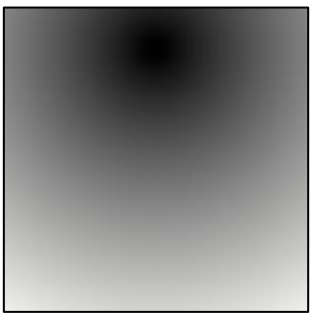


Figura 94: distance field di B

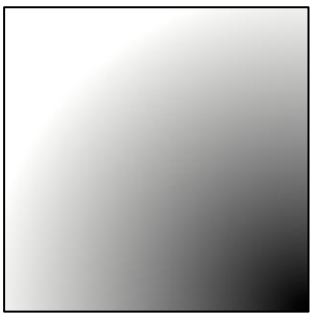


Figura 95: distance field di C

Successivamente si possono confrontare tra di loro i campi a coppie, valutando il minimo fra i due.

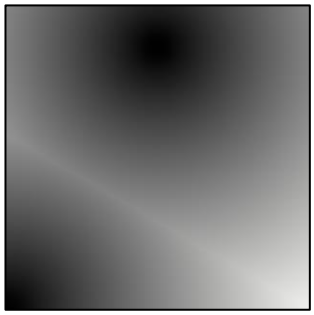


Figura 96: min(dfA, dfB)

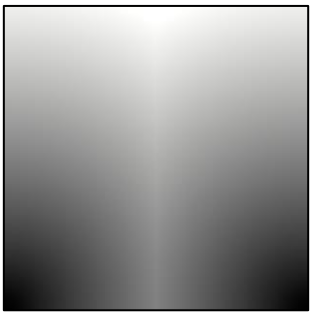


Figura 97: min(dfA, dfC)

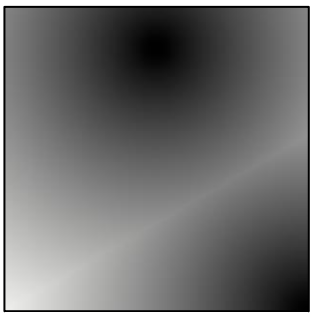


Figura 98: min(dfB, dfC)

A partire da questi distance field è possibile iniziare ad effettuare tutti i calcoli necessari alla generazione del pattern. Il primo passo successivo è quello di verificare quando i valori della mappa ottenuta per due punti sono minori di quelli del distance field del punto complementare.

Volendo anche gestire lo spessore del bordo degli esagoni, a questo punto è utile introdurre una variabile apposita detta Mortar Size. Sommando questa

variabile ai distance field relativi ai punti complementari prima di verificare la diseguaglianza si può ottenere la variazione desiderata.

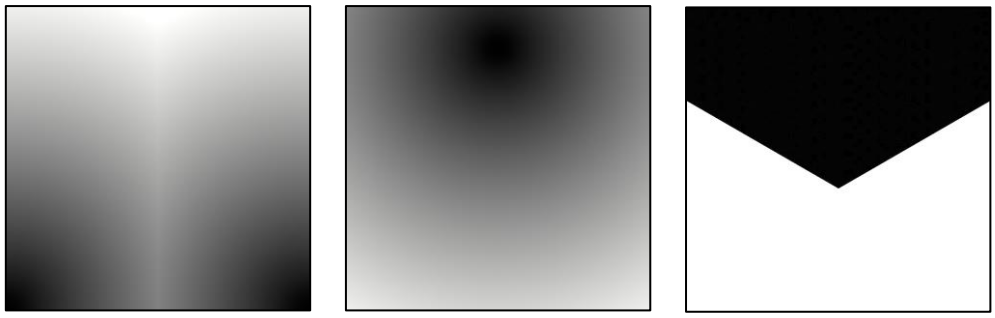


Figura 99: in ordine dfAC, dfB e il risultato di dfAC<dfB

Figura 100: alcune varianti ottenibili variando il Mortar Size, in particolare 0.0, 0.5, 1.0

Non resta che eseguire il procedimento per ognuno dei punti e moltiplicarle fra di loro ottenendo così il risultato desiderato.

Figura 101: le tre mappe, la risultante della loro moltiplicazione e l'aggiunta delle barre laterali

La cellula base del pattern è quasi pronta per essere replicata, non resta che aggiungere le due barrette ai lati superiori. Questo può essere facilmente realizzato sfruttando quanto appreso precedentemente: generando un pattern a righe verticali e successivamente mascherandolo con le ultime mappe ottenute. Il risultato finale è mostrato nella *Figura 101*.

Coordinate specchiate

Nel presente paragrafo si affronterà come replicare, traslare e specchiare quanto ottenuto, mantenendo una variabilità di scala globale e del singolo esagono (tramite Mortar Size).

Essendo l'elemento di base definito per $x \in [0, 1]$ e $y \in \left[0, \frac{\sqrt{3}}{2}\right]$, sarà necessario che la griglia generata sia di un aspect ratio diverso da quello quadrato. Verranno utilizzati quindi il metodo di replicazione che sfrutta la funzione modulo.

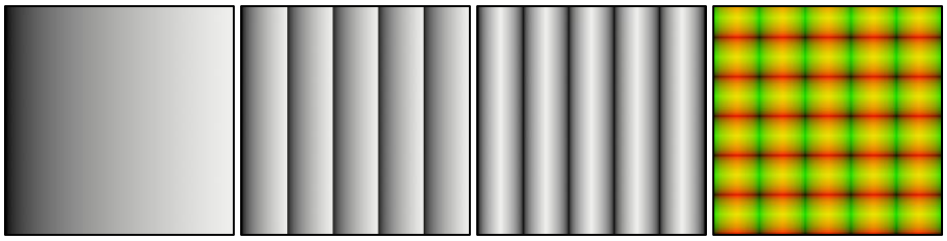


Figura 102: generazione delle coordinate lungo x e coordinate complete

Con le coordinate così ottenute è possibile finalmente generare il pattern esagonale. Nelle immagini successive sono riportati degli esempi sulle possibilità che si hanno variando il Mortar Size.

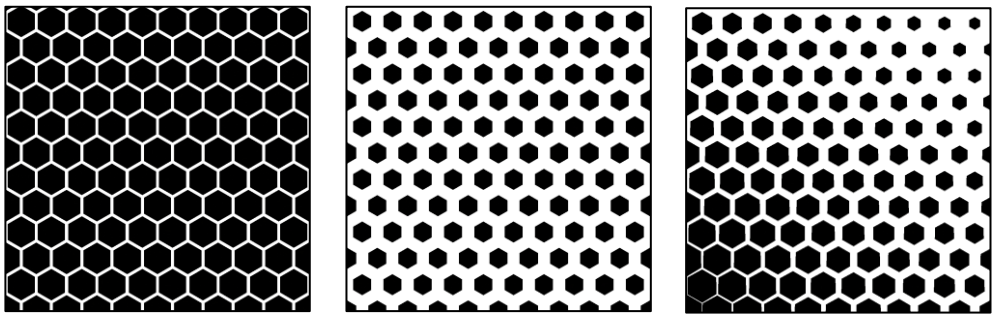


Figura 103: risultato finale

Colori casuali

L'ultimo passaggio da affrontare è l'implementazione della variazione casuale dei colori degli esagoni.

Al fine di raggiungere lo scopo è necessario numerare le celle tramite una coppia di valori identificativa. Sebbene questo possa essere intuitivamente

semplice, a causa della tecnica utilizzata ciò diventa particolarmente arduo. In particolare, bisogna fare in modo di preservare la continuità della numerazione da una cella all'altra ricordandosi però che l'elemento di partenza contiene al suo interno tre parti di esagoni diversi.

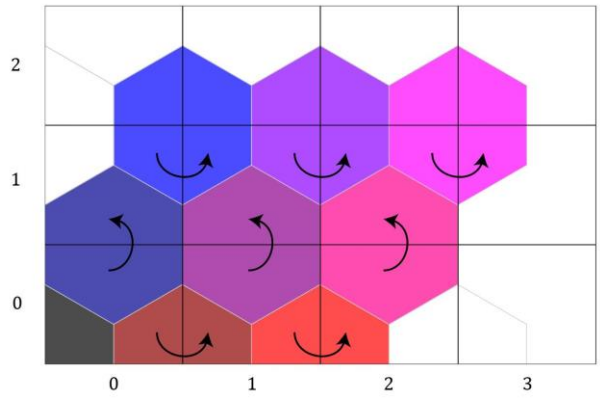


Figura 104: continuità da imporre attraverso le celle

Il primo passo da effettuare è creare una maschera che definisca quali colonne e righe sono specchiate e quali no. Una volta ottenute queste maschere sarà possibile incrementare in modo conseguente il valore di ogni cella basandosi sui distance field calcolati durante la prima fase. È possibile infine utilizzare i valori ottenuti come coordinate per un noise di Perlin, il risultato sarà quello desiderato.

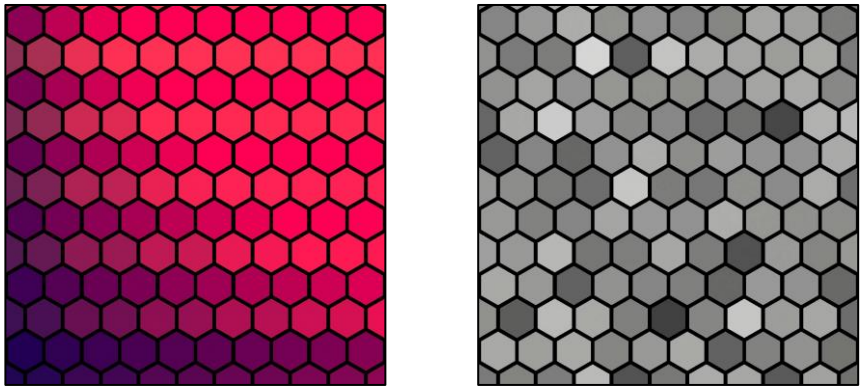


Figura 106: le coordinate e il risultato finale

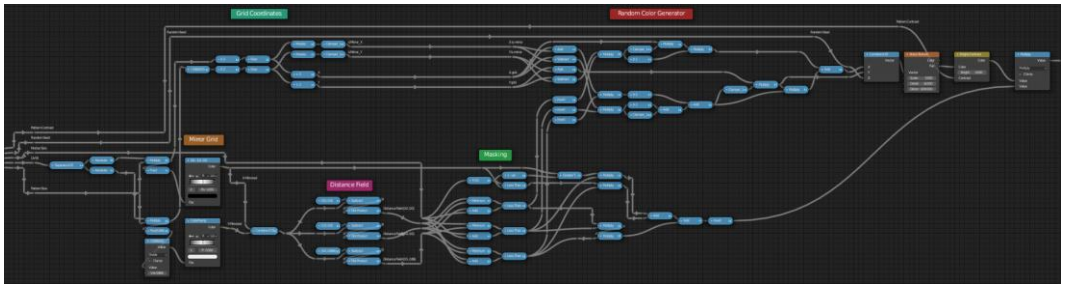


Figura 105: l'implementazione su Blender

5.1.4 Osservazioni

Dagli esperimenti svolti si sono riusciti ad ottenere tutti i pattern ricercati. Il passaggio successivo nella ricerca è stato quello di effettuare delle prove sul modo migliore per sfruttare i pattern: in che zone applicarli, con che intensità e nel caso con quale proprietà influenzarla, come scalarli, etc. Le prove sono state innumerevoli, per sinteticità di trattazione, ed essendo di fatto un processo empirico, non verranno riportate in questa sede. Nella prossima sezione si passerà dunque all'analisi di quello che effettivamente è diventato il setup di produzione al momento di stesura di questo documento.

5.2 Setup di produzione

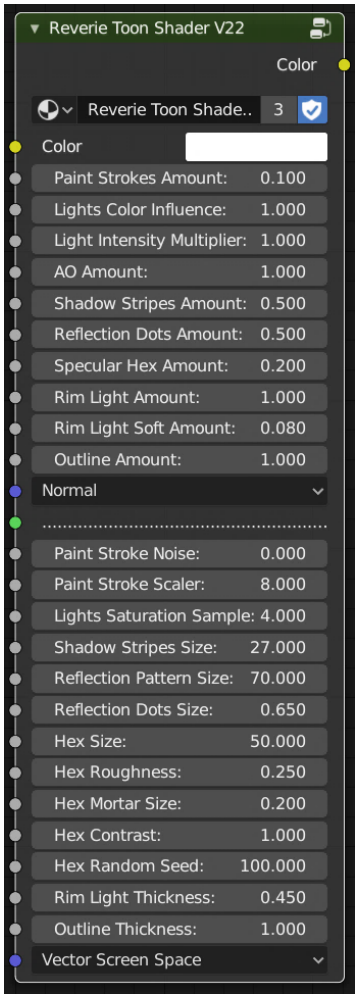


Figura 107: Reverie Shader V22

L'obiettivo finale era quello di fornire uno strumento intuitivo, utilizzabile anche da un artista che non avesse particolari capacità tecniche legate allo shading. Si è dunque provveduto ad incapsulare quanto precedentemente illustrato in blocchi funzione separati e infine si è realizzato un macro-blocco che esponesse tutte le variabili d'interesse per lo shader artist. Nell'immagine a fianco si può osservare il risultato della ricerca svolta. Il nodo è diviso in due sezioni, quella superiore è dedicata all'intensità delle singole componenti (*Amount*), in quella inferiore invece si trovano tutti i parametri necessari alla gestione dell'aspetto estetico: scala, forma, random seed, contrasto, samples, dimensione dei singoli elementi del pattern, coordinate screen space, etc.

Il risultato è stato un compromesso fra flessibilità e complessità. Il problema più grande emerge nel momento in cui quando è necessario effettuare delle modifiche ai valori dei

pattern propagandole a tutti i materiali in scena. Questo può accadere molto spesso, anche solo quando si effettua un cambio di inquadratura. Dover cambiare parametri manualmente ad ogni istanza dello shader diventa quindi molto dispendioso, soprattutto perché i materiali in scena raggiungono spesso l'ordine delle centinaia. D'altra parte, spesso non c'è la necessità di cambiare i valori assegnati allo shader, basta che questi rimangano al loro valore di default. Si è quindi pensato di realizzare un nodo complementare che avesse il ruolo sia di contenitore dei valori di default che di pannello di controllo globale.

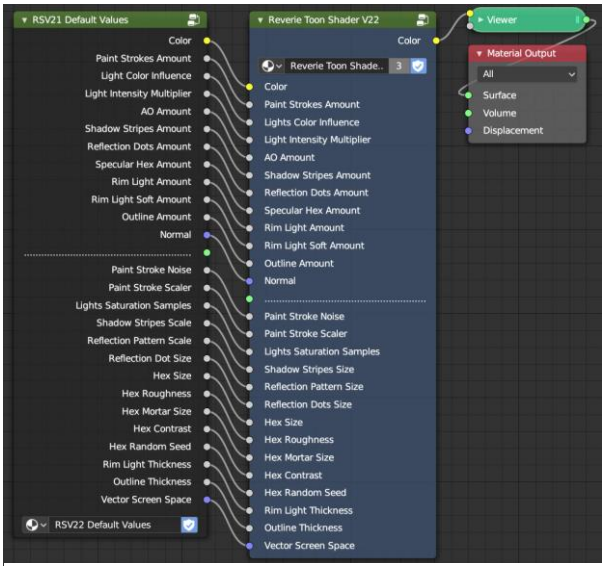


Figura 108: connessione del nodo Default Values

Il nodo presenta lo stesso numero di socket, nominati allo stesso modo rispetto a quelli che devono ricevere i valori. I collegamenti sono illustrati nella *Figura 108*. All'interno del nodo Default Values è stata realizzata invece un'interfaccia dove esporre e poter accedere comodamente i valori di default. Il risultato finale si può osservare nella figura sottostante.

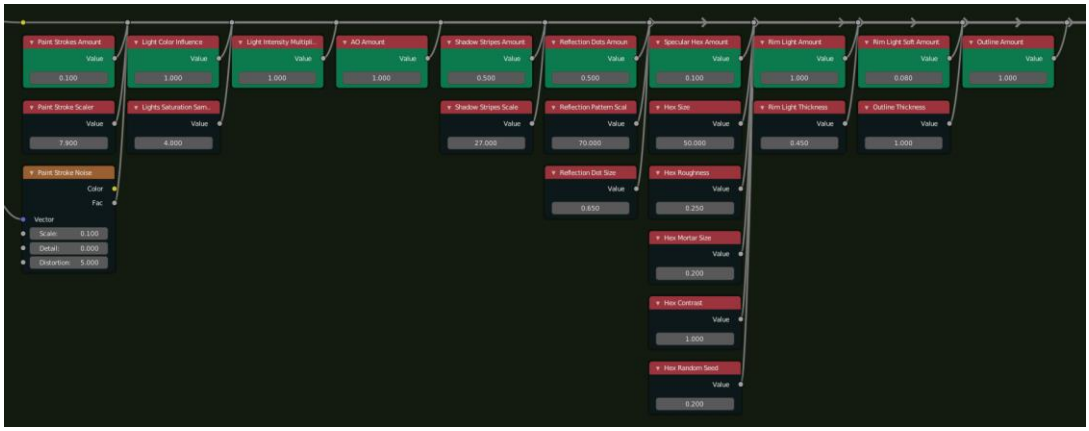


Figura 109: pannello di controllo

Questo processo di riordinamento di nodi è stato fondamentale al fine di fornire uno strumento utilizzabile. L'alternativa sarebbe stata costringere l'utente a districarsi nel mezzo di tutti i blocchi funzionali.

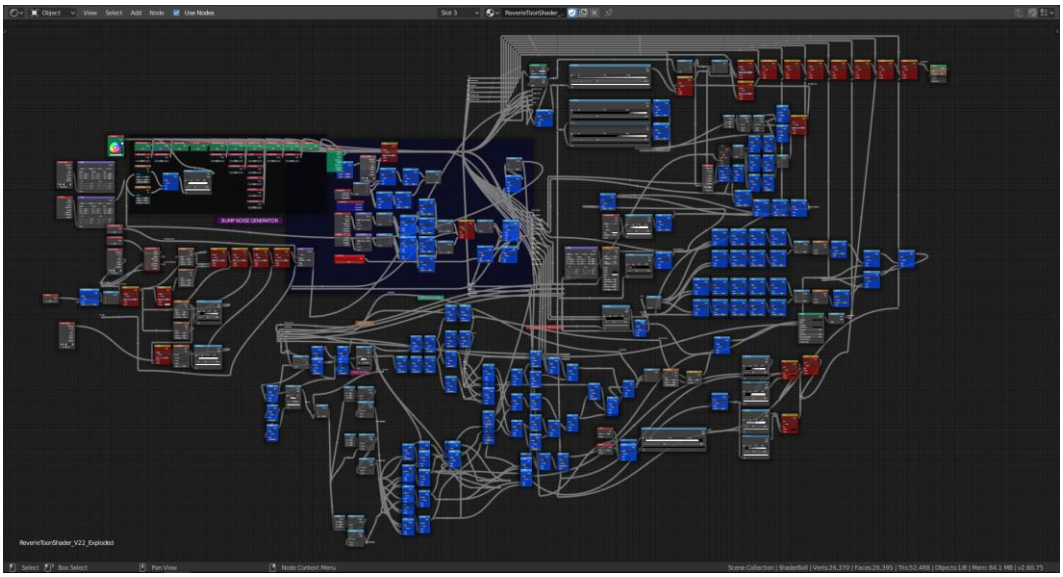
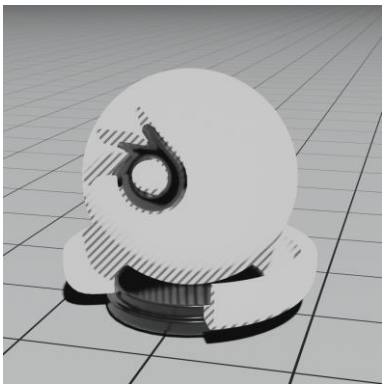


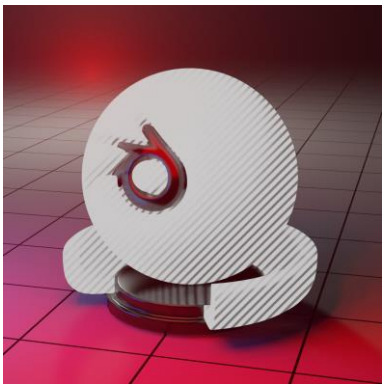
Figura 110: tutti i nodi che compongono lo shader

Nei prossimi paragrafi verranno quindi brevemente presi in esame i singoli blocchi funzionali implementati, confrontando la resa con i setup di luci finora utilizzati

5.2.1 Shadow

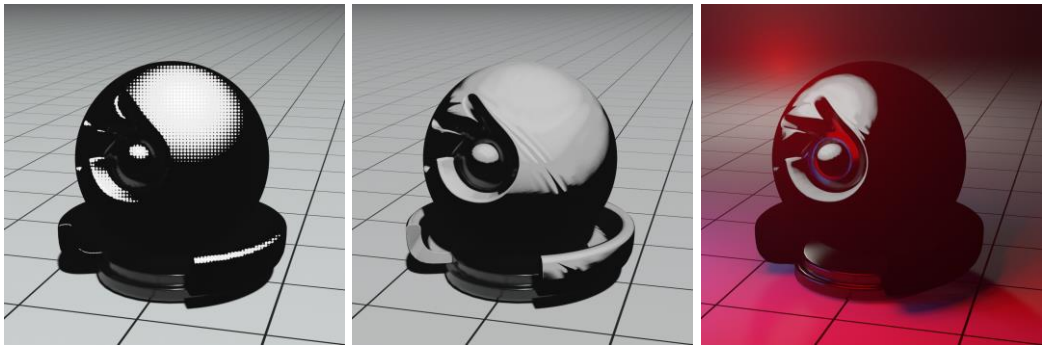
Utilizzando i valori ottenuti a partire da un Diffuse standard come illustrato nel paragrafo SHADER TO RGB si è mascherato un retino a righe generato in screen space.





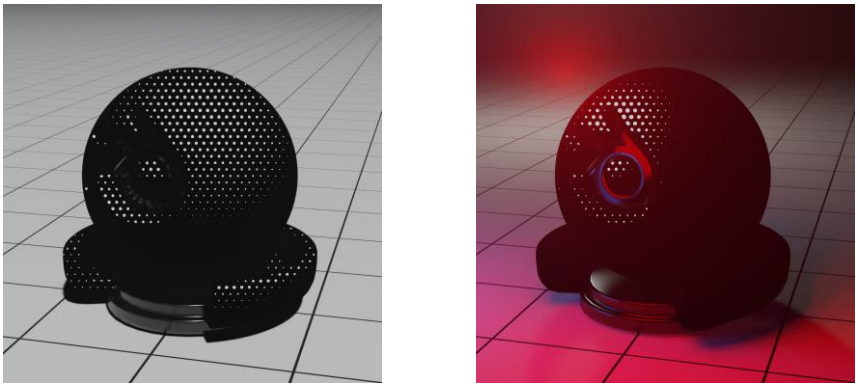
5.2.2 Paint Strokes Extra

Volendo arricchire la texture nelle zone illuminate, si è deciso di aggiungere una funzione dedicata. Questa utilizza un retino a punti con griglia quadrata generato in screen space, scalando i singoli punti proporzionalmente al valore di illuminazione. È stata inoltre implementata la possibilità di aggiungere un rumore personalizzato alle coordinate utilizzate, con l'obiettivo di trasformare all'occorrenza il pattern regolare in qualcosa di diverso. Le varianti ottenibili sono mostrate nelle immagini che seguono.



5.2.3 Glossy

Utilizzando lo stesso principio del paragrafo precedente, in corrispondenza dei punti di alte luci dovuti alla componente glossy, si è aggiunta una griglia triangolare di punti.

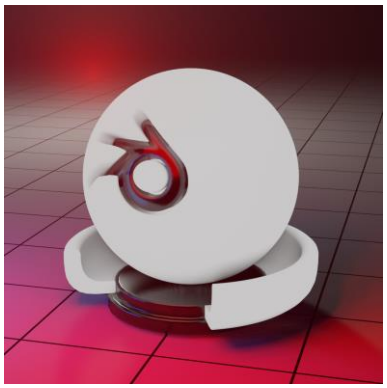


5.2.4 Light Color Influence

Uno dei problemi emersi durante l'analisi era la perdita dell'informazione del colore delle luci. L'opzione per cui si è optato è stata quella di separare le informazione di Hue e Saturation ottenute dallo stesso Diffuse Shader dal quale si generano le maschere per tutte le altre funzioni. Successivamente si procede ad una quantizzazione della saturazione per poi riunire le varie componenti, ottenendo così un effetto di discretizzazione sui colore, aumentando l'effetto cartoon. Il risultato viene infine unito, tramite il metodo di fusione Overlay, al colore o texture di base dell'oggetto. Nelle immagini che seguono sono nell'ordine presentati: lo shader senza Light Color Influence, la saturazione campionata con 2 step, la saturazione campionata con 4 step.

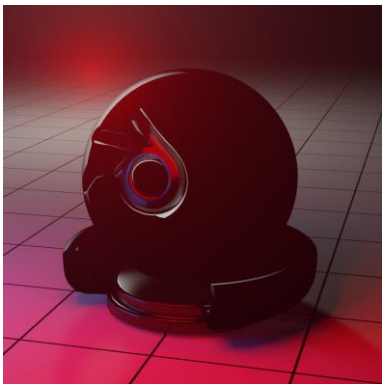
5.2.5 Ambient Occlusion

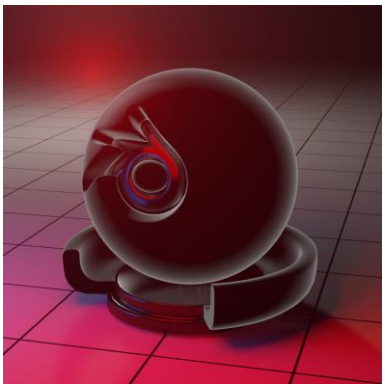
Al fine di ridurre il processo di compositing, si è deciso di includere nello shader l'aggiunta della componente di Ambient Occlusion.



5.2.6 Rim light

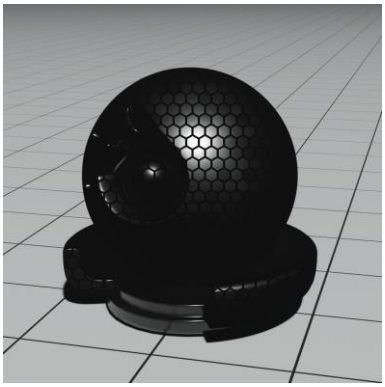
Utilizzando la tecnica spiegata al paragrafo FRESNEL INPUT sulle rim light in object space, si sono aggiunti due livelli di effetto: una rim light netta, ottenuta con clamping del Fresnel e limitandolo alle zone di luce, e una più morbida che avvolge l'intero oggetto. Le due vengono poi sommate.

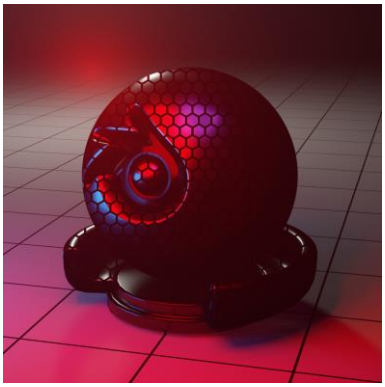




5.2.7 Specular

Volendo aggiungere qualcosa di unico e caratteristico, si è pensato di sfruttare il pattern esagonale illustrato al paragrafo GRIGLIA ESAGONALE per mettere in risalto i riflessi speculari. Per poter ottenere il risultato desiderato si è fatto uso dello shader Specular BSDF, in grado di restituire solo la componente speculare.



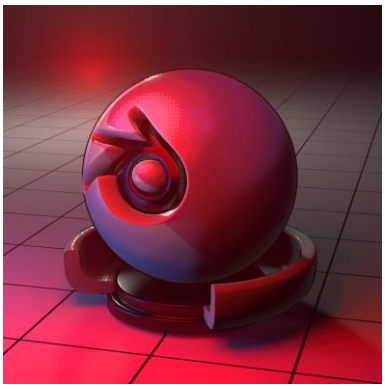


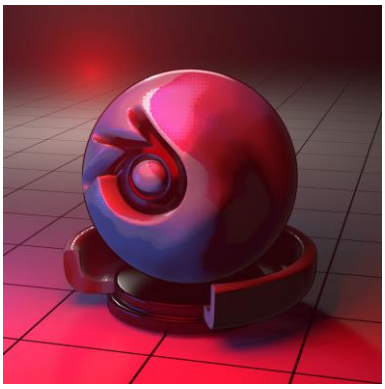
5.2.8 Bump Noise

Al fine di ottenere delle variazioni interessanti, si è deciso di implementare la possibilità di collegare delle mappe di normali allo shader Diffuse mascherante. Le possibilità sono pressoché infinite, nel caso della produzione si è realizzato un preset basato sull'unione di diversi rumori di Perlin e Voronoi generati in object space. Le variazioni ottenibili riescono a dare delle variazioni particolarmente interessanti, soprattutto se l'obiettivo è quello di dare un'impressione di texture a pennellata.

L'unico problema emerge quando l'oggetto viene animato, in quanto le variazioni introdotte nella superficie possono essere percepite come fastidiose.

Di seguito è mostrato un confronto fra una versione senza e con bump map.





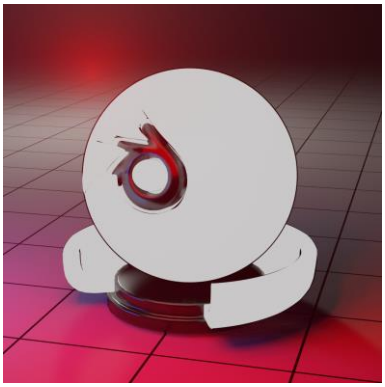
5.2.9 Outline e Freestyle

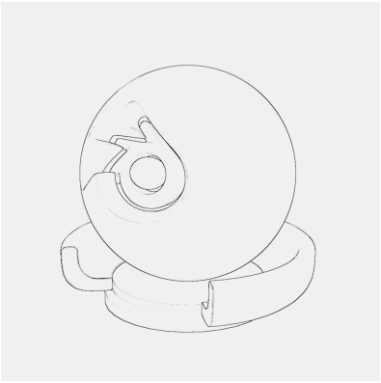
Sfruttando lo stesso principio utilizzato per la generazione della rim light, si è implementato un contorno generato in object space. L'influenza dell'effetto è gestibile interamente dall'interfaccia dello shader. Questo però non è particolarmente stabile a causa delle limitazioni dovute all'approccio che fa uso del Fresnel. Si è comunque deciso di mantenere l'opzione per le situazioni in cui fosse servito aggiungere maggiore enfasi sui contorni.

Cercando una soluzione alternativa, si è quindi deciso di recuperare la tecnica utilizzata per calcolare le maschere di displacement nel paragrafo Edge Displacement, sfruttandola questa volta per ottenere un tratto molto simile a

quello che si può avere con il Freestyle, senza il tempo di rendering aggiuntivo.

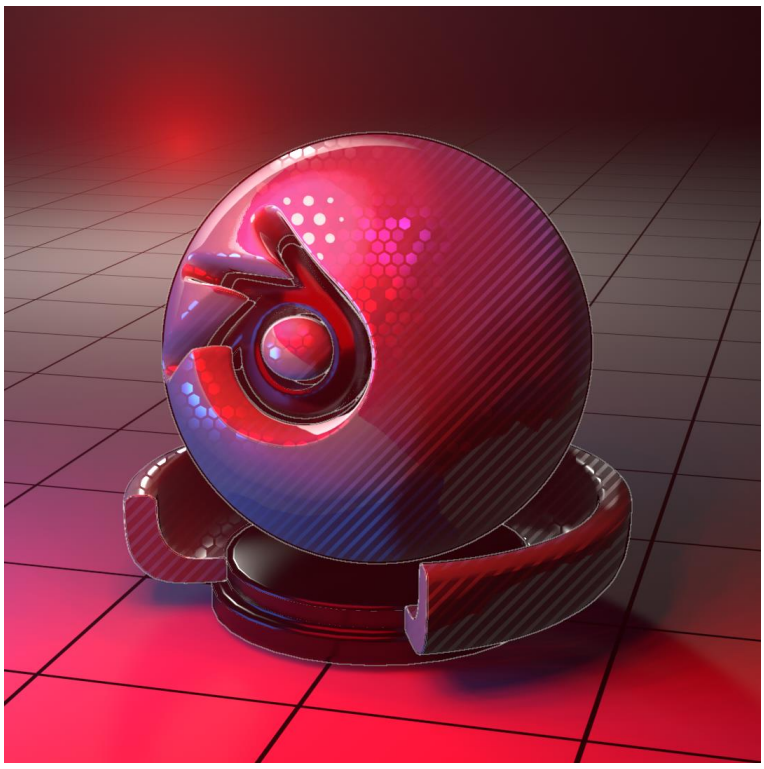
Di seguito sono confrontate la versione ottenuta in object space e quella in screen space.





5.3 Risultati finali di shading

Viene riportato di seguito il risultato finale. Nel prossimo capitolo verranno affrontate le fasi di illuminazione, si avrà quindi modo di vedere l'algoritmo applicato a delle scene del progetto.



6 Lighting

Il contesto all'interno del quale si svolge la storia narrata in Reverie Dawnfall ha richiesto uno studio piuttosto approfondito fin dalle fasi di preproduzione. Il pianeta che ospita le vicende dei protagonisti è perennemente illuminato da una stella (simile al sole) sempre bassa sull'orizzonte. Questo aspetto caratterizza fortemente il tipo di illuminazione degli ambienti (sia interni che esterni): si ha una forte direzionalità della luce che imprime tinte decise in tutti i punti che raggiunge direttamente. Inoltre, si ottengono aree fortemente contrastate nelle zone non illuminate direttamente.

Figura 111: Concept art città

Nel resto del capitolo, saranno analizzate le scelte inerenti all'illuminazione ambiente per ambiente, mostrando risultati intermedi e analizzando i parametri che hanno permesso di convergere verso il risultato finale.

6.1 Stanza di Nadya

La stanza di Nadya è un ambiente chiuso con un'unica grande finestra illuminata direttamente dal "sole". Quest'ultima rappresenta la fonte di luce principale dell'ambiente, coadiuvata da sorgenti secondarie artificiali quali i

pannelli luminosi “intelligenti” presenti sul soffitto e le strisce di led sulle pareti e sul pavimento.
La costruzione dell’illuminazione si è basata su alcuni punti chiave:

- Ottenere un fascio di luce netto e ben definito in ingresso dalla finestra che si mescolasse in maniera efficace con le tinte delle luci artificiali interne alla stanza.
- Creare un gradiente cromatico in direzione trasversale alla stanza che partisse e terminasse con le tinte “fredde” dei led, passando per la componente “calda” della luce proveniente dall’esterno, zona all’interno della quale compare per la prima volta la protagonista.
- Creare un forte contrasto di luminosità in direzione longitudinale alla stanza: la massima luminosità si ha in corrispondenza della finestra e l’intensità diminuisce gradualmente man mano che ci si allontana da lei, arrivando a zone d’ombra abbastanza marcate nelle quali emergono solo i pulsanti luminosi sulle pareti.

La sorgente principale è data da un *Sun* che punta diretto sul pavimento della stanza: è la componente “naturale” della luce, identificata dalla sua tinta particolarmente calda. A contrastarlo sono state posizionate delle Area Light in corrispondenza dei monitor negli angoli della stanza. A completare la componente fredda ed artificiale dell’illuminazione sono state distribuite lungo la stanza delle strisce di led che aiutano ad esaltare la geometria peculiare degli arredi e rivelano una serie di dettagli che rendono più “vivo” l’ambiente.

Nella seguente immagine il risultato ottenuto è stato abbastanza soddisfacente, con l’unico difetto della perdita di una serie di dettagli nella parte alta della stanza.

Figura 112: stanza di Nadya illuminata

Si è deciso dunque di effettuare il bake della luce indiretta con evidenti migliorie nelle zone di interesse come riportato nel render successivo.

Figura 113: stanza di Nadya illuminata e con il bake della luce indiretta

6.2 Strada

La strada ha senza dubbio rappresentato una delle sfide più interessanti dal punto di vista dell'illuminazione.

Le ambientazioni cittadine sono un cliché del genere Cyberpunk e dunque

fin dalla fase di concept si sono cercate soluzioni che fossero pertinenti al canone "imposto" dal genere, ma che allo stesso tempo permettessero di esaltare la cifra stilistica della serie.
Si è deciso di incominciare a sperimentare possibili soluzioni a partire da alcuni elementi chiave definiti in fase di concept:

- Il pianeta è perennemente illuminato da una luce crepuscolare, bassa sull'orizzonte.
- Le strade cittadine che la protagonista percorre sono molto strette con palazzi alti ed imponenti. Inoltre, si tratta di strade tendenzialmente affollate.
- Le facciate dei palazzi sono, per la quasi totalità della loro altezza, ricoperti da insegne luminose di ogni genere.

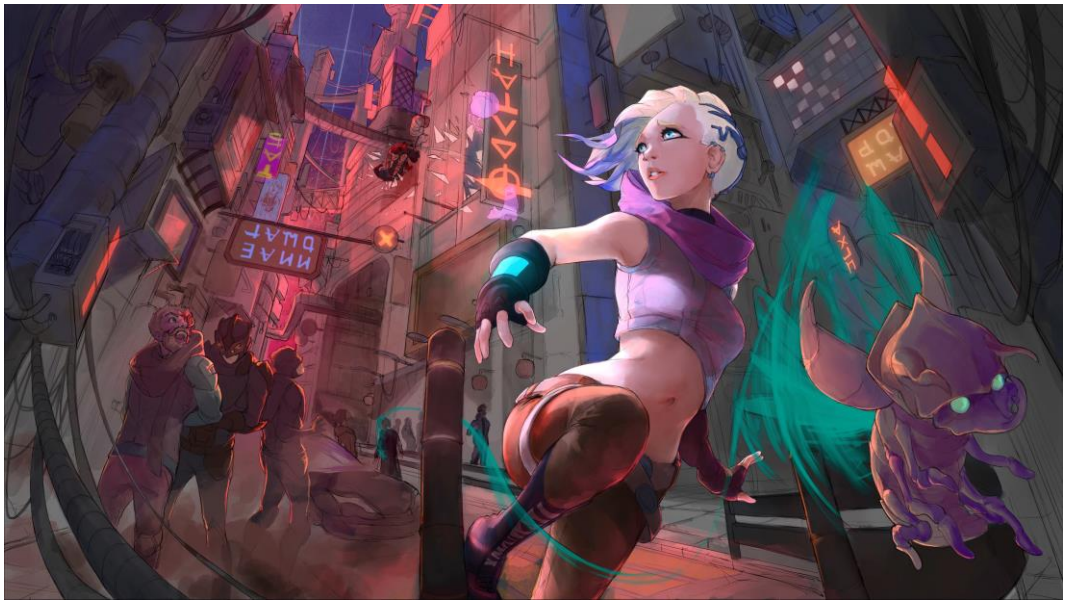


Figura 114: concept art delle strade cittadine e di Nadya

Nelle scene del teaser di Reverie Dawnfall sono presenti diverse inquadratura ambientate tra le strade cittadine, ciascuna con le proprie esigenze estetiche e narrative.

Nella quasi totalità delle inquadrature in strada, la protagonista è seguita dalla camera e deve destreggiarsi in mezzo alla folla.

L'obiettivo perseguito per la sequenza in strada era proprio quello di trasmettere la caoticità dell'ambiente. Proprio per questo si è deciso di inserire un significativo numero di pannelli luminosi di tutti i colori lungo la via. Di fatto, questi ultimi rappresentano una delle due sorgenti principali di luce dell'intera strada.
L'altra componente fondamentale è data dal sole: essendo "basso" filtra di taglio tra i palazzi e colpisce la parte di strada verso la quale Nadya è diretta. La posizione del sole, unita all'introduzione di variazioni cromatiche dei pannelli luminosi permette di far percepire nettamente allo spettatore la direzione verso la quale la protagonista corre.

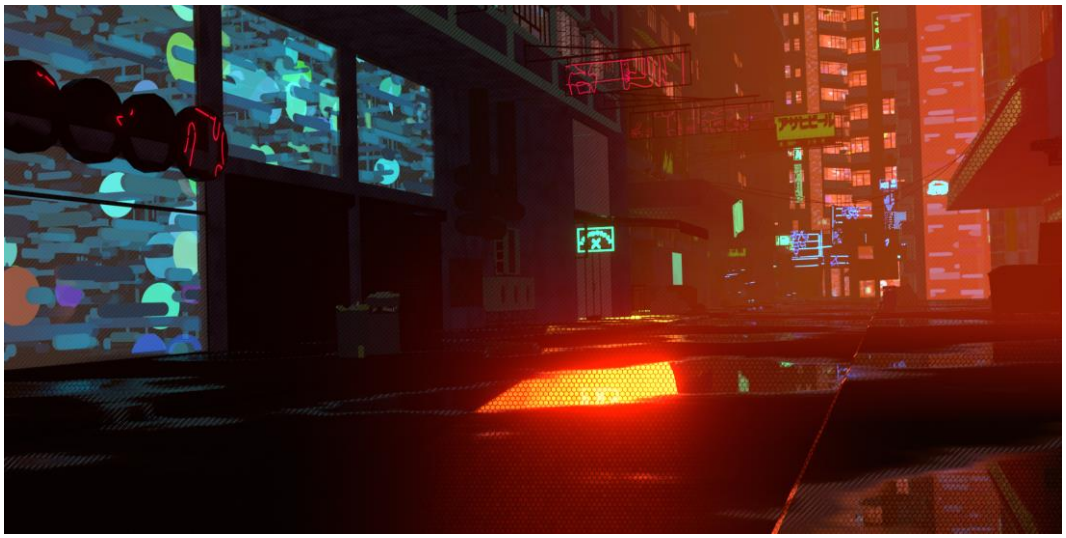


Figura 115: strada illuminata dal sole

Altra componente fondamentale per lo sviluppo dell'estetica delle ambientazioni cittadine è stata l'introduzione di consistenti variazioni cromatiche e di luminosità. Per poter garantire un tale risultato si è optato per l'adozione di una serie di *Area light* per i seguenti motivi:

- Fonti di luce con alta intensità e limitata influenza hanno permesso di creare forti contrasti volti a far emergere alcune peculiarità dei modelli degli edifici e definire i profili della folla che popola la strada.
- Il tipo di distribuzione di luce garantita dalle Area light permette di esaltare le caratteristiche dello shader: aree di alta intensità luminosa

fanno emergere i pattern 2D (si veda Pattern 2D). Inoltre, il parametro di *Light Color Influence* proprio dei materiali permette di caratterizzare cromaticamente zone ben circoscritte.

Figura 116: strada illuminata dal sole e dalle Area light

Infine, per ottenere il risultato desiderato si è dovuto valutare opportunamente l'uso del parametro *bloom* in fase di render. La brillantezza e la saturazione delle luci, unite allo shader volumetrico del "mondo" ha permesso di garantire l'atmosfera nebbiosa tipica del genere cyberpunk.

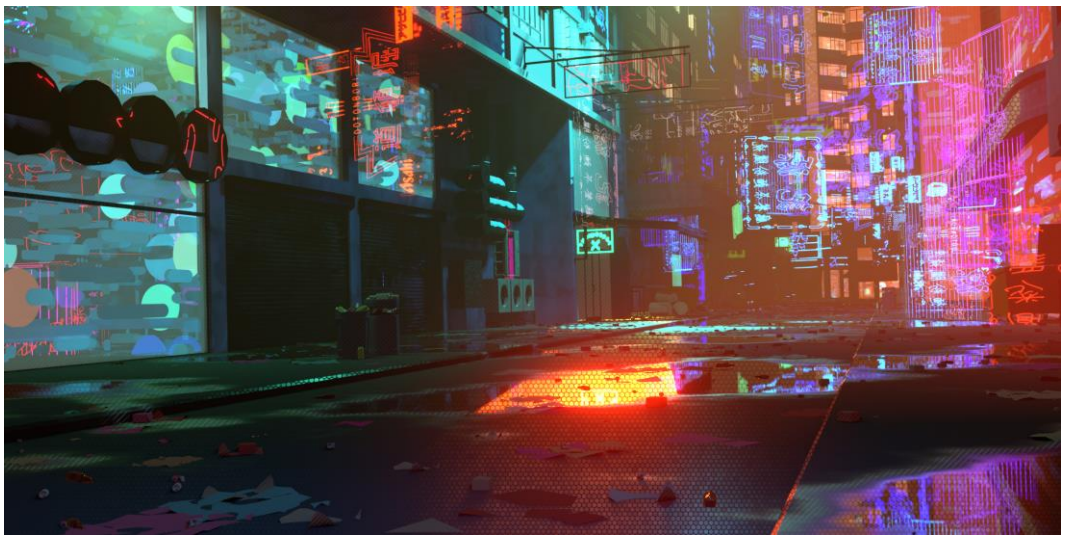


Figura 117: versione definitiva della strada

6.3 Void

Il Cliff è un ambiente esterno, completamente estraneo a tutto quanto visto fino a quel punto. Quando la protagonista raggiunge il Cliff è stupefatta ed a tratti spaesata. Era dunque necessario trovare un espediente (dal punto di vista estetico e dell'illuminazione) che creasse uno stacco abbastanza netto, tanto da causare anche nello spettatore degli attimi di disorientamento.
Gli obiettivi principali da raggiungere erano i seguenti:

- Realizzare un ambiente che fosse profondamente diverso da quanto visto fino a quel momento, tuttavia mantenendo i paradigmi estetici e stilistici principali definiti per la serie.
- Cercare di introdurre un ambiente più "asciutto", in contrasto con l'atmosfera "nebbiosa" delle ambientazioni precedenti.
- Introdurre dei nuovi elementi (oggetti con "shader volumetrici") che aggiungessero valore estetico e rappresentassero una nuova sfida tecnica dal punto di vista della produzione.

La sorgente di luce principale è sempre un *Sun* frontale alla protagonista. Per le inquadrature più strette (quelle che prevedono primi piani o inquadrature a mezzo busto su Nadya) sono state inserite una *fill light* della stessa tinta del Sun principale ed una *back light* che riprende le tinte della nebulosa nello spazio.

Per quanto riguarda il fondale, si è deciso di mantenerlo il più "piatto" possibile (marcando il nero profondo) per poter esaltare i dettagli delle stelle e della nebulosa.

Proprio la nebulosa ha rappresentato la sfida più interessante. Fin dai primi tentativi si era intuito che il modo migliore per realizzarla fosse introdurre un volume che contenesse tutta la scena ed attribuirgli uno shader volumetrico. È stata applicata al volume una noise texture 3D (in modo che ad ogni elementino di volume fosse attribuito un valore tra "bianco" e "nero") e ne

sono state estratte solo alcune zone con una color ramp (in modo da creare dei vuoti).

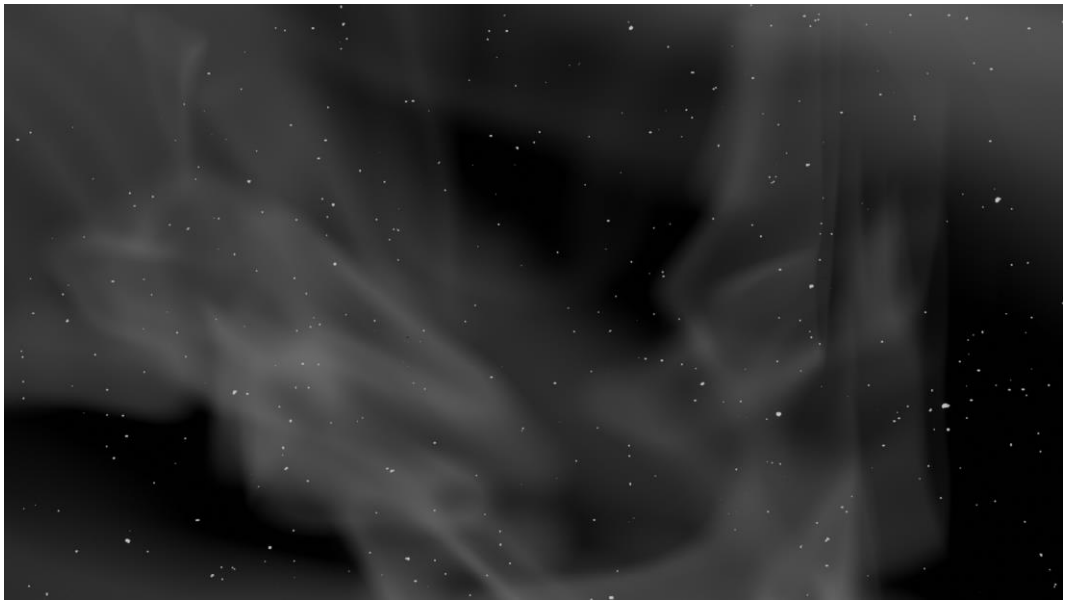


Figura 118: noise volumetrico in bianco e nero

Successivamente è stata applicata un'altra color ramp per attribuire alle zone meno dense un colore più scuro ed alle zone più dense un colore più acceso, con una tinta leggermente diversa.

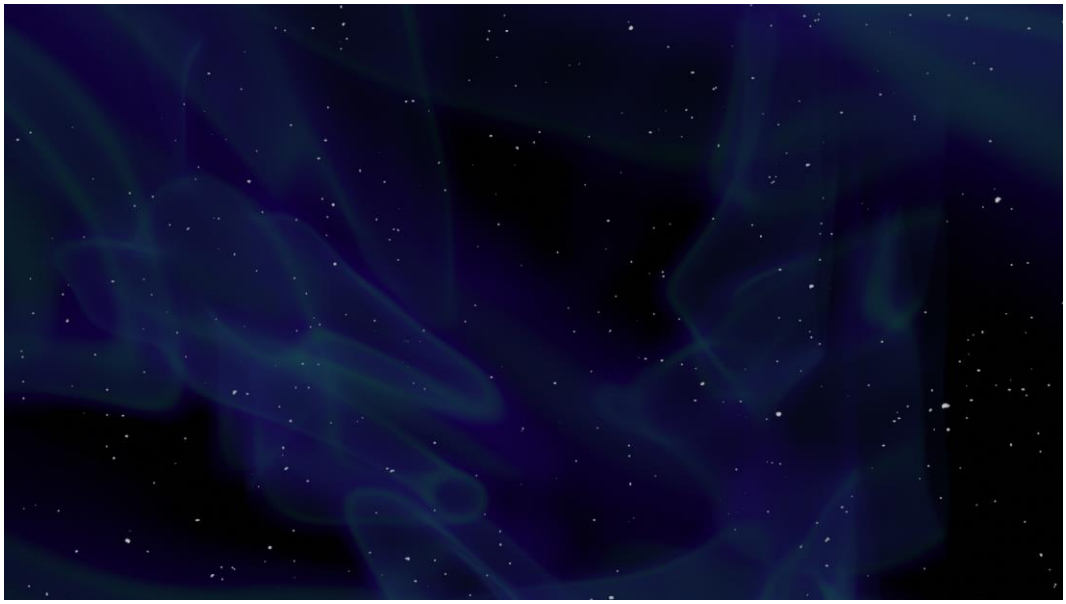


Figura 119: nebulosa colorata senza pattern

Il risultato è parso abbastanza soddisfacente ma non coerente con il resto dell'estetica dettata dai paradigmi dello shader utilizzato per tutti i materiali. Si è deciso dunque di applicare al volume un *dot pattern* che fungesse da ulteriore filtro per la densità della nebulosa. Il risultato è stato sorprendentemente positivo: regolando i diversi parametri in gioco, si è riusciti ad ottenere un tipo di variazione che ha permesso di esaltare le zone più scure di vuoto accentandole con un pattern coerente con il resto dei materiali.

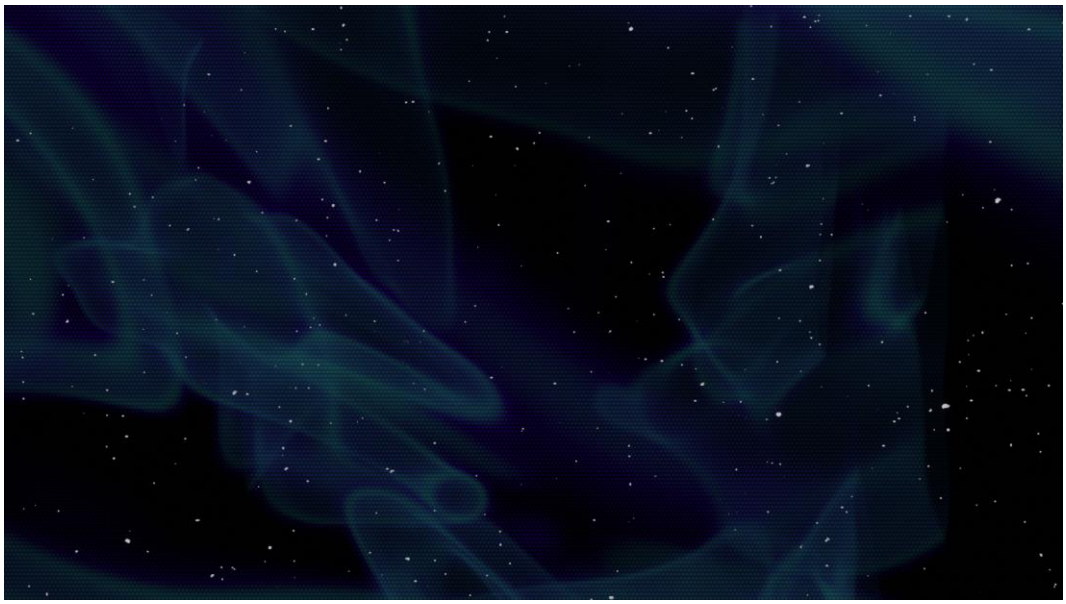
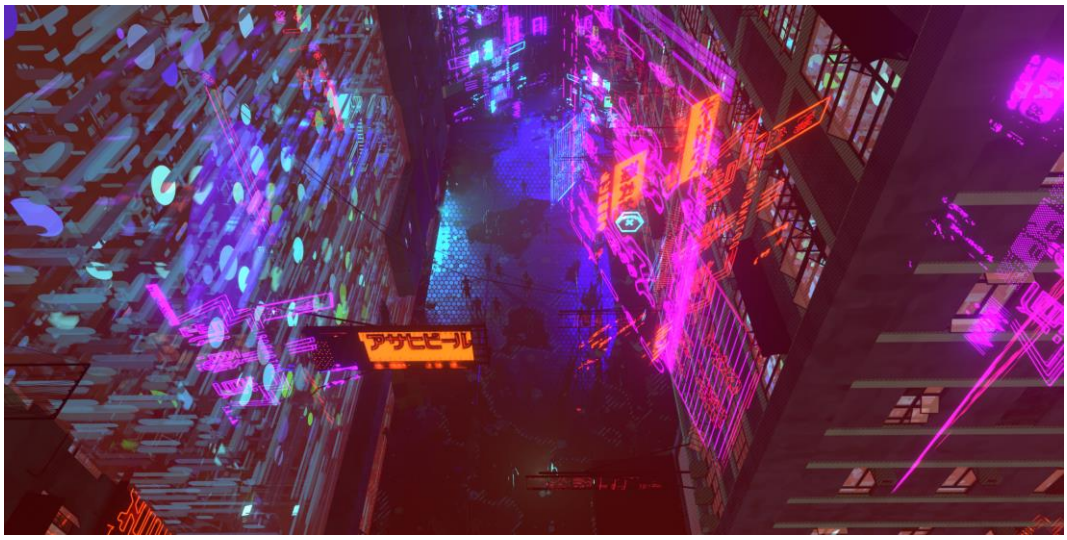


Figura 120: nebulosa colorata con dot pattern

7 Risultati Finali



8 Conclusioni e prospettive future

Nel presente capitolo saranno analizzati, per ciascun settore trattato, i risultati ottenuti durante il periodo di ricerca del presente lavoro di tesi con un occhio di riguardo anche alle possibili ed auspicabili direzioni da prendere per eventuali sviluppi futuri per la produzione della serie Reverie Dawnfall e progetti affini.

8.1 Light Studio add-on

Come ampiamente trattato nel Light Studio add-on, i risultati ottenuti sono stati estremamente soddisfacenti. L'impatto del plugin sulla produzione (ed in generale anche sul lavoro di chiunque si approcci a realizzare progetti in Blender) è stato piuttosto importante e si possono considerare raggiunti la maggior parte degli obiettivi che ci si era posti in fase di progettazione e di sviluppo.

La fase di test dopo le prime release candidate è stata fondamentale: ha permesso di far emergere un limite che avrebbe potuto inficiare l'intera efficacia di Light Studio. Come riportato, è stato possibile implementare una modifica risolutiva prima della conclusione della presente trattazione ma non è stato possibile effettuare una nuova campagna di test.

In ogni caso il progetto Light Studio resta un progetto con ancora tante potenzialità da esprimere, un progetto che si presta molto facilmente a future espansioni e integrazioni.
Nella fattispecie, trattandosi di un add-on che mira a rendere più efficiente il lavoro in Blender, verrà certamente presa in considerazione la possibilità di sviluppare l'intera interfaccia di Light Studio all'interno di un contesto di Blender esclusivamente dedicato (spazio simile a quello riservato al *Node editor*).

Inoltre, si valuterà, a valle dell'introduzione e del test di alcune piccole migliorie, l'introduzione di Light Studio all'interno di canali di diffusione "ufficiali" di plugin per Blender come ad esempio il sito https://blendermarket.com/.

8.2 Lighting

Anche per quanto riguarda l'illuminazione degli ambienti per Reverie Dawnfall ci sono importanti margini di sviluppo, sia dal punto di vista artistico, sia dal punto di vista tecnico.

Per quanto riguarda l'estetica del prodotto, i tentativi fatti fino a questo punto hanno portato a risultati piuttosto soddisfacenti: il paradigma estetico dettato in fase di concept è stato fedelmente rappresentato e successivamente arricchito con degli accorgimenti che si sarebbero potuti ottenere esclusivamente con una adeguata ricerca e sperimentazione degli strumenti che Blender, ed il suo motore di rendering EEVEE, mettono a disposizione.
Restano però aperte possibilità molto interessanti grazie ad una tecnologia relativamente giovane: RTX.
L'RTX è di fatto il supporto che le schede grafiche di ultima generazione mettono a disposizione per il rendering in tempo reale con motori raytracing (si veda *Real time vs path tracing*). Questa tecnologia ha già rappresentato una svolta piuttosto significativa nel mondo dei videogiochi e della pre-visualizzazione, consentendo di sviluppare prodotti decisamente più affascinanti sia in ambito fotorealistico che non fotorealistico. Resta quindi da scoprire se e come Blender deciderà di sfruttare la prerogativa delle schede grafiche di ultima generazione e come la integrerà all'interno dei propri motori di rendering.

8.3 Shading

I risultati finali possono definirsi soddisfacenti, lo shader ha risposto bene alle diverse prove a cui è stato sottoposto ed è esteticamente molto simile all'aspetto di un fumetto. L'aggiunta di un outline, generato in compositing sfruttando le pass delle normali, e la scelta di animare i personaggi in passo due ha sicuramente aiutato molto nel raggiungere l'obiettivo desiderato.

L'interfaccia messa a disposizione all'utente ha reso possibile facile e veloce propagare le impostazioni predefinite a tutti i materiali. La realizzazione di un piccolo rig esterno ha anche reso possibile l'animazione delle proprietà dello shader, aumentando le possibilità in fase di lighting.

In futuro sono sicuramente diverse le migliorie apportabili al prodotto con lo scopo di migliorarlo. In primo luogo, l'applicazione di un filtro di quantizzazione alle normali dei modelli, come fatto in Spiderman – Into the Spiderverse, può aiutare a rendere più evidente la componente di pennellate procedurali.

Per quanto riguarda la generazione dell'outline, al momento è in fase di sviluppo un motore di rendering interamente dedicato alla generazione realtime dei contorni: il LANPR. Quest'ultimo non ha però una roadmap definita, al momento della scrittura i risultati sono promettenti ma decisamente instabili per una pipeline di produzione.

Un ulteriore ambito interessante da esplorare è sicuramente quello del nuovo ambiente di animazione ibrido 2D/3D dato dal *Grease Pencil*. Con questo è possibile animare e realizzare forme 2D come overlay e pattern extra, andando così ad aggiungere elementi tipici del linguaggio del fumetto.

Un'ultima considerazione va fatta sulla poca possibilità di azione in fase di compositing: in futuro, con l'aggiunta delle funzionalità di sovrascrittura dei

materiali in EEVEE, si potranno esportare dei passi di render specifici ai singoli pattern, aggiungendo la possibilità di applicare o evidenziare i pattern in zone specifiche dell'immagine.

In conclusione, il Look Dev del progetto è giunto ad un livello adeguato ad una pipeline di produzione, raggiungendo gli obiettivi prefissati. In futuro sono comunque molte le strade che si possono percorrere per affinare il risultato ottenuto.

9 Ringraziamenti

Ehi lettore furbetto! So che hai scorso rapidamente tutte le sudate pagine precedenti per arrivare fin qui, curioso di pettegolezzi. Tuttavia, non mi offenderò perché mi conosco abbastanza bene da sapere che se sei qui a curiosare ti ho ammorbato a sufficienza con il contenuto dei capitoli precedenti. Quindi mettiti comodo e goditi queste righe con la speranza che ti facciano battere un po' più forte il cuore, restituendoti i battiti che (probabilmente) ti ho rubato tre anni fa quando ho deciso di iniziare questa avventura.

Trovare le parole giuste per riassumere il percorso fatto negli ultimi tre anni non è stato assolutamente semplice, ma ho cercato le più adatte a farlo.

L'immagine più chiara che ho trovato per descriverlo è quella del viaggio (sì ok, probabilmente non la più geniale o la più originale). Non un viaggio qualunque, ma un viaggio ben riuscito e certamente non ancora concluso. Sì, perché quando arrivi in una nuova stanza e la prospettiva dal tuo letto non è quella rassicurante che hai da una vita, ma una ignota, il primo pensiero che hai è quello di essere in viaggio, fiducioso che a breve sarai di nuovo a casa. Poi i giorni passano e impari a capire che quella che per te era la prospettiva rassicurante della tua stanzetta non era che una delle infinite possibili. E allora pian piano capisci che una prospettiva non ti basta più, che più ne sperimenti e più la tua mente si apre, più impari a mettere a fuoco ciò che conta.

Arrivati a questo punto del viaggio credo sia il caso di fare un piccolo bilancio, guardare indietro e ringraziare chi ha contribuito a renderlo speciale.

Ringrazio Ele per essere la migliore compagna di viaggio che potesse capitarmi, di quelli con i quali si sente tutto all'unisono, per i quali ogni parola (anche quelle scritte) rischia di essere banale.

Ringrazio i miei genitori per aver sponsorizzato e supportato il viaggio dopo aver visto brillare una scintilla di convinzione nei miei occhi quando anche io non avevo garanzie e nonostante i sacrifici necessari.

Ringrazio Annamaria, i miei nonni, i miei zii ed i miei cugini per la stima incondizionata che nei sentieri più nebbiosi del percorso è fondamentale per restare lucidi e prendere la decisione giusta.

Ringrazio Caterina, Checco, Eleonora, Gianvito, Lorenzo, Marco e Sara che da viaggiatori (più o meno esperti) mi hanno insegnato a viaggiare ogni oltre confine e che le distanze fisiche non contano quasi nulla se il mondo lo si guarda con gli stessi occhi.

Ringrazio Antonio, Chiara, Mattia e Rebecca perché in ogni viaggio c'è bisogno di trovare rifugio di tanto in tanto in un posto accogliente, nel quale sentirsi a casa. Un posto dove sciogliere ogni tensione, ricaricare le batterie e ripartire rifocillati.

Ringrazio Jack e tutto il gruppo vacanze Btriaskrkk perché con loro ho scoperto che viaggiatori e passanti lungo la strada possono rivelarsi preziosi compagni di viaggio e perché quello che abbiamo condiviso negli ultimi mesi va oltre la vicinanza dei nomi nei titoli di coda e nei momenti di studio condiviso.

Andrea Lorusso

Ciao, sono io, il vostro Jack.

Credo che il modo migliore per chiudere questo documento sia abbandonare la scrittura impersonale che quasi mi ha portato all'alienazione. Sarò breve e conciso a differenza del Capitano (bugia).

Durante questi cinque anni di percorso al Politecnico tante cose sono cambiate, tante sono le persone che ho conosciuto, quelle che sono entrate e uscite dalla mia vita. Quindi secondo una distribuzione di Gauss-Balma ci tengo a ringraziare tutta Btriaskrkk per l'amore che mi ha dato, il divano di casa Cino-Martin che mi ha permesso di dormire e il piumone della FraCap che mi ha tenuto al caldo anche il cuore, le torri capovolte che mi ha augurato FraAlbert con i suoi oroscopi, le recensioni delle pizzette dell'Eurospin di Gennaro, i piatti con texture granito di Giorgina, il muscolo duro affilato dei becchi di Bitritto e i ponti di NicoZ, il pulito di Gian ma anche i gelati al sugo che porta, il pesantissimo ma emblematico orologio di Brunetto, i manichini e la ciabatta abbandonata a Lido di Pomposa di MUugo, i porza traku della parlata del Commodoro, il limbo di Gabri e il Fresnel, i consigli calcistici del Capitano che sono tanto veri in campo quanto nella vita, l'Argentina che è tanto lontana ma non abbastanza da separare due persone, il Messico e tutte le splendide persone che tanto mi hanno dato in così poco tempo, i pareri di Apaxlegomenon, i Cinemini tutti quanti, gli amici ma solo su facebook, Mario Ambrogetti, "Una chat di poche parole" e la pro loco "Gli amici del meme", la ricchezza che mi ha regalato Velia, Nicola per averci ospitati a pane e cinese in casa sua, Ele per le giornate che escono, Usseglio e tutto ciò che ne è derivato, la Nuii per l'averci portato le noci macadamia australiane e il caramello salato, tutti quelli che hanno lavorato a Reverie, il buco misterioso di Ila, le shapekey di Ste, meganico, le immagini profilo a tema sul gruppo Telegram, gli esagoni, il Robilant e il campetto dietro (e gli scarpini di Gabri), OpenProcessing e quelli che mi vedono lì, *@_iamlab* su Instagram, Cubetto per tenere freschi gli Spritz delle Panche, le persone molto alte, tutta la crew del set di

Nightfall, i wurstel, la pazienza di tutti nel sopportare le mie lamentele, i capelli, i keyframe, la Blender Foundation e i bug, Otto, il risotto e le polpette di Imperia Edvige, HiTech per avermi tenuto compagnia, le responsabilità e la burocrazia, la pizza di bassa qualità, i flauti suonati da chi non è in grado di farlo e la Bose per la sua voce sintetizzata nelle casse Bluetooth.

Per concludere volevo ringraziare tutta la mia famiglia. Grazie a mamma e papà che mi hanno sempre spinto ad essere creativo e non mi hanno mai fatto mancare nulla perché ciò potesse avvenire. Grazie a nonno Romano per avermi insegnato come funziona un chiavistello e come si innestano delle piante, perché da lì è nato tutto.

Giacomo Balma

10 Bibliografia e sitografia

- Carrozzino Marcello. *Modellazione procedurale*. SSSUP, 2007.
- Rogers, David; Earnshaw, Rae. *Computer Graphics Techniques: Theory and Practice*. Springer, 2001.
- F. Kenton Musgrave, Darwyn Peachey. *Texturing and Modeling: a Procedural Approach*, Morgan Kaufmann Pub, 2003.
- Ramin Zahed. *Spider-Man: Into the Spider-Verse: the Art of the Movie*, Titan Books, 2018.

- https://thebookofshaders.com/ - The Book of Shaders
- http://robin.studio/reverie/ReveriePressbook.pdf - Robin Studio
- https://www.blender.org/ - Blender
- https://devtalk.blender.org/ - Blender Developer Talk
- http://www.rendereverything.com/tiles-shader-blender/ - Render Everything
- http://www.neilblevins.com/cg_education/cg_education.htm - Neil Blevins
- https://www.fxguide.com/fxfeatured/why-spider-verse-has-the-most-inventive-visuals-youll-see-this-year/ - FXguide

Sommario