Introduction to TouchDesigner (Italian)

nVoid

Introduction to TouchDesigner (Italian)

nVoid

Questo libro è in vendita presso http://leanpub.com/introductiontotouchdesigner-it

Questa versione è stata pubblicata il 2019-04-04

Indice

Introduzione

Prefazione

Tra l'artista ed il suo strumento si inseriscono innumerevoli tecniche e conoscenze, che contribuiscono perché il primo diventi più fluido e meglio preparato per le sfide della produzione. Quando mi è giunta voce del progetto del libro su TouchDesigner di Elburz, ho pensato: "Mi piacerebbe sapere quello che sa lui!". La profonda conoscenza di Elburz relativamente a TouchDesigner è stata costruita da numerose fonti – attraverso l'aiuto e la conversazione con migliaia di utenti sui forum, revisionando progetti dalle scadenze impellenti, lavorando insieme a Derivative utilizzando le versioni beta di TouchDesigner, e grazie a curiosità genuina, mista alla sua vita parallela da trombonista e VJ.

Il libro è un ottimo compagno di tastiera o di comodino: esplora i concetti, le tecniche e le metodologie dietro TouchDesigner - cosa che io non ho mai avuto l'opportunità di proporre personalmente, e che sono molto contento Elburz abbia voluto presentare qui. Perciò, costruite i vostri CHOP con questi piacevoli consigli, scoprendo gemme di informazione e approfondimenti in tutto il libro (che può essere letto sia dall'inizio alla fine, oppure sfogliando a caso tra le pagine, come spesso mi ritrovo a fare io).

Molte grazie ad Elburz per la sua iniziativa volta ad arricchire la community di TouchDesigner. Sono sicuro che scatenerà una forte reazione a catena.

Greg Hermanovic Fondatore Derivative

Prefazione dell'Autore

Lo scopo di questo libro è duplice:

- insegnare i fondamenti di TouchDesigner 088;
- creare una risorsa per principianti guidata dalla community.

Il primo obiettivo è immediatamente comprensibile. Osserveremo i vari elementi dell'interfaccia utente, discuteremo gli Operatori e le famiglie di Operatori, esploreremo i flussi logici, le ottimizzazioni dei Network, le interfacce per le performance, la gestione della visualizzazione, eccetera. Verranno spiegati e mostrati gli elementi fondamentali di TouchDesigner, e saranno affrontati preventivamente molti problemi comuni.

Dopo la parte scritta, impareremo come approcciare la soluzione di un problema con progetti di esempio e video tutorial, posti alla fine del libro. Queste lezioni saranno pratiche, in quanto assembleremo partendo da zero alcuni progetti e moduli utili.

Il secondo obiettivo di questo libro è leggermente più gravoso sulla community. Crediamo veramente nella comunità perciò, invece di vendere questo libro, abbiamo voluto che tutti fossero liberi di accedervi gratuitamente (testo, video e file di esempio). Abbiamo voluto portare quest'idea un passo oltre, consentendo non solo di fruire dei contenuti del libro, ma anche permettendo libero accesso ai mattoni che ne costituiscono il nucleo stesso.

Ciò significa che tutti possono sentirsi liberi di aggiungere, commentare, cambiare, porre in evidenza o cancellare le risorse di questo libro. Tutto il testo è stato scritto utilizzando LaTeX, ed il codice per compilare il libro, i file dei progetti, gli esempi, i grafici, i video, e qualsiasi altra cosa presente, saranno tutti raccolti su GitHub, sempre a titolo gratuito (sotto licenza Creative Commons) perché chiunque possa accedere, scaricare, condividere e costruirci sopra.

A scopo di controllo qualità, a tutti sarà consentito modificare il repository, ma noi vaglieremo tutti i cambiamenti prima di integrare gli aggiornamenti nel libro principale. In questo modo un nuovo utente dovrà cercare un singolo PDF, o un solo link di download, per ricevere la conoscenza più aggiornata della community.

Speriamo vivamente che la comunità si appassioni e ci aiuti a creare la guida per principianti definitiva!

Come scritto all'inizio, sapere cos'è questo libro è tanto importante quanto sapere cosa non è. Questo libro non è un manuale di riferimento degli Operatori. Non copriremo ogni parametro o gli utilizzi di ogni Operatore. Questo libro non è pensato per sostituire la Wiki di Derivative come riferimento principale; useremo ed impareremo solamente ciò di cui abbiamo bisogno. Questo libro non intende inoltre rimpiazzare gli utilissimi utenti ed i componenti presenti nel forum.

Infine, questa risorsa è un tributo ai tanti programmatori TouchDesigner e allo staff di Derivative che, sia sul forum sia di persona, hanno aiutato noi tutti ad arrivare dove siamo. Speriamo che questa tradizione duri tanto quanto TouchDesigner.

Elburz Sorkhabi & nVoid Art-Tech Limited

Cos'è TouchDesigner?

Questa è una domanda a cui, in fase iniziale, molti utenti dedicano le loro energie, cercando una risposta. Può sorprendere quanto tempo occorra per creare ed eseguire compiti semplici. Può stupire il fatto che molto di questo tempo sia speso per implementare funzionalità già presenti nativamente in altri pacchetti software. Quindi, cos'è TouchDesigner? La risposta è semplice: TouchDesigner è un linguaggio di programmazione visuale, basato su nodi.

Partendo dall'aspetto probabilmente più importante della descrizione, TouchDesigner è un linguaggio di programmazione. TouchDesigner non è un'applicazione che, appena aperta, è già predisposta

per eseguire azioni che in altri programmi sono immediate. TouchDesigner è un ambiente dotato di estrema profondità, insieme a molte potenziali insidie. Con un po' di pratica e di esperienza, molte richieste possono essere soddisfatte velocemente. Grazie allo scopo del libro di creare moduli riutilizzabili, aumenterà di molto la velocità in cui la tela bianca diventerà un progetto finito. Ciò non toglie il fatto che TouchDesigner sia anche un linguaggio di programmazione. Molti compiti richiederanno la dovuta dedizione, sia come tempo impiegato, sia in quanto a fatica. Molti progetti causeranno parecchi grattacapi, e tutti avranno bisogno di abilità basilari di problem-solving.

Il secondo punto della descrizione è che TouchDesigner è basato sui nodi. Ciò significa che, invece di aprire un documento di testo e digitare il codice riga dopo riga, per creare le applicazioni si utilizza l'interfaccia grafica di TouchDesigner, partendo dai nodi. Ogni nodo, in TouchDesigner chiamato Operatore, compie un'azione specifica, piccola e granulare. Per eseguire compiti complessi, una manciata di nodi lavoreranno insieme. Per inviare informazioni tra i nodi, i loro input ed output devono essere collegati. Esistono molti linguaggi di programmazione basati sui nodi, come Max/MSP di Cycling74, ma ciò che contraddistingue TouchDesigner è la sua natura visuale.

Tutto all'interno di TouchDesigner ha una controparte visuale. Tutti gli Operatori hanno un viewer. Ogni cosa, sia essa testo, dati di controllo, audio, video, o altro, è visualizzata attraverso ognuna delle operazioni compiute. Questo è diverso da ogni linguaggio di programmazione tradizionale, anche quelli basati sui nodi, ma è ciò che rende TouchDesigner un ambiente fantastico in cui lavorare. Imparare a compiere compiti complessi è di gran lunga semplificato dall'abilità di visualizzare i passaggi coinvolti, ad ogni passo del processo.

Accedere al libro

Ci sono molti modi per accedere a "Introduzione a TouchDesigner 088". Da quando il libro è stato convertito usando Gitbooks, è possibile scaricare il PDF, l'epub o la versione mobi del libro dal link seguente:

https://www.gitbook.com/book/nvoid/introduction-to-touchdesigner/details

I file di esempio si trovano nella cartella `TouchDesigner Example Files` del repository di GitHub, che può essere clonato e scaricato qui:

https://github.com/nVoid/Introduction-to-touchdesigner

I video tutorial HD sono disponibili in un unico canale Vimeo. Tutti i file possono essere scaricati da Vimeo per fruirne offline. Il link è il seguente:

https://vimeo.com/channels/845218

Si può inoltre scaricare il codice sorgente Markdown da questo GitHub. Siete liberi di cambiare, correggere, aggiungere, eliminare e modificare il testo, creando un branch dal file originale. Quando avete terminato, inviate i vostri cambiamenti e saranno controllati da un nostro amministratore, e uniti nel ramo principale. Sarete quindi aggiunti alla lista dei contributori nel capitolo "Contributors" del libro, insieme con il vostro apporto.

Per altre informazioni generali riguardo le varie risorse, visitate il sito http://book.nvoid.com

Compilare il libro

Diversamente dalla versione precedente in LaTeX, non è necessario alcun ambiente di compilazione per contribuire al libro. Potete usare il normale linguaggio Markdown, e quando avrete terminato, e i vostri cambiamenti saranno stati accettati nel repository, Leanpub creerà automaticamente una nuova versione web, PDF, epub e mobi del libro.

Attribuzione e Licenza

Note sulla traduzione italiana

Questa traduzione si basa sul manuale “Introduction to TouchDesigner”, scritto da Elburz Sorkhabi e disponibile sul sito: http://book.nvoid.com/

Ho cercato di rimanere il più fedele possibile all’originale, mantenendo i termini inglesi per la parte legata all’interfaccia e alla costruzione dei network, e tentando di tradurre dignitosamente il resto del testo. Ogni errore nella traduzione è da imputare solamente a me.

Il manuale è relativo alla versione 088 del software, quindi potrebbero essere presenti leggere differenze rispetto agli aggiornamenti più recenti.

Un grazie a Elburz Sorkhabi e a Matthew Hedlin per il supporto durante la fase finale della stesura di questa versione, e per il tempo speso nella sua impaginazione.

Infine, un doveroso ringraziamento alla community italiana di TouchDesigner, ed in particolare al gruppo Facebook “TouchDesigner Italian Community Group”, sempre disponibile per confrontarsi su questioni tecniche e non solo, e che si è prestato con utili consigli e correzioni per migliorare questo manuale. Il risultato è un tentativo di restituire parte di quello che voi avete dato a me, sperando di alimentare ulteriormente la diffusione del creative coding e di TouchDesigner nel nostro paese.

Per qualsiasi commento o correzione, o semplicemente per scambiare due chiacchiere, non esitate a contattarmi direttamente sui social (Facebook e Instagram: Deltacut) o via mail (info@deltacut.it).

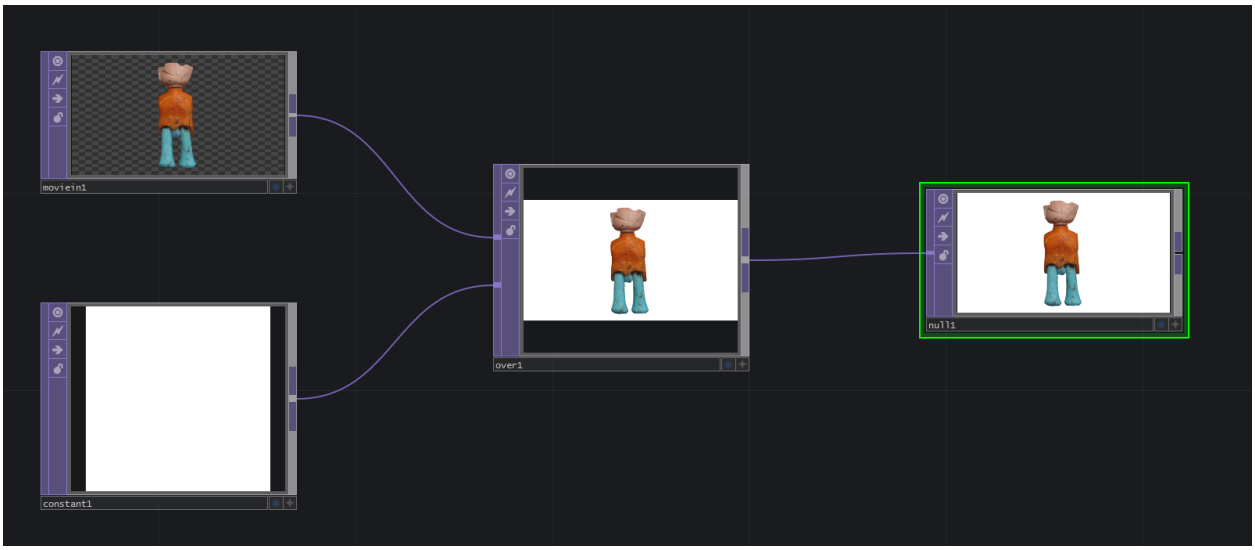
Davide Santini // Deltacut // www.deltacut.it

1 Basi

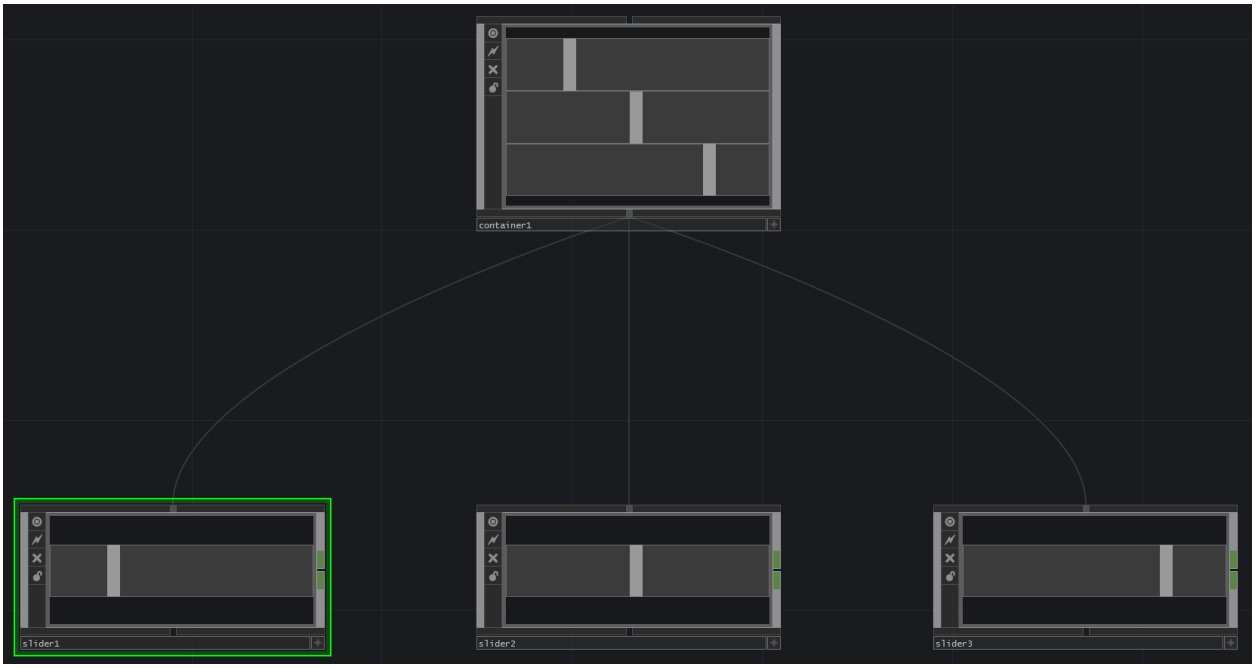
1.1 Flusso del Segnale e Collegamenti

L'operazione più importante in TouchDesigner è il collegamento degli Operatori. Tutti i progetti sono costituiti semplicemente da gruppi di Operatori collegati assieme. Ogni Operatore, da solo, compie un'azione molto specifica, ma quando più Operatori sono collegati insieme in un Network, essi possono risolvere problemi estremamente complessi.

Tutti i dati in TouchDesigner scorrono da sinistra a destra. Ogni input sarà sempre sul lato sinistro del suo Operatore, mentre gli output saranno sul suo lato destro. Gli ingressi e le uscite saranno inoltre ordinate dalla prima all'ultima partendo dall'alto. Nella figura di esempio successiva, seguite i due segnali partendo da sinistra. Mentre scorrono da sinistra a destra, sono composti uno sopra l'altro:



È interessante notare come i Componenti seguano lo stesso schema dei segnali degli Operatori, scorrendo da sinistra a destra, ma siano anche capaci di instaurare relazioni di genitore e figlio, che evolvono invece dall'alto in basso. Il Componente in cima alla catena del segnale è il genitore, i Componenti sottostanti sono i suoi figli, sotto di questi ci sono i figli dei figli e così via. Nell'esempio seguente è mostrata una piccola interfaccia utente, composta da alcuni slider. In questo esempio i COMP "Slider" sono i figli del COMP "Container", che è il genitore:


container1
slider1
slider2
slider3

1.2 Creare gli Operatori

La finestra di dialogo per creare gli Operatori può essere raggiunta in molti modi, ognuno adatto a determinati momenti e situazioni. I due metodi più semplici quando si vogliono creare Operatori da zero sono: premere il tasto “Tab” sulla tastiera, o fare doppio click sullo sfondo del Network.

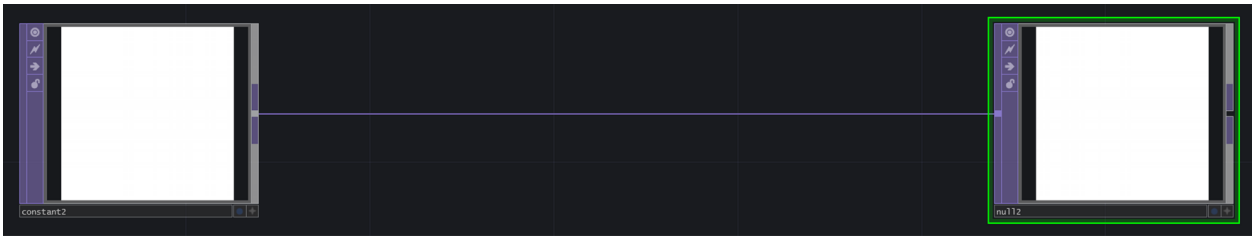
Quando si lavora con catene di Operatori già esistenti, si possono aggiungere Operatori con due metodi molto semplici. Il primo consiste nel fare click con il tasto destro sull’input oppure sull’output di un Operatore. In questo modo l’Operatore scelto verrà aggiunto, già collegato, immediatamente prima dell’ingresso o subito dopo l’uscita. Ciò si rivela utile specialmente per inserire l’Operatore direttamente in una catena preesistente.

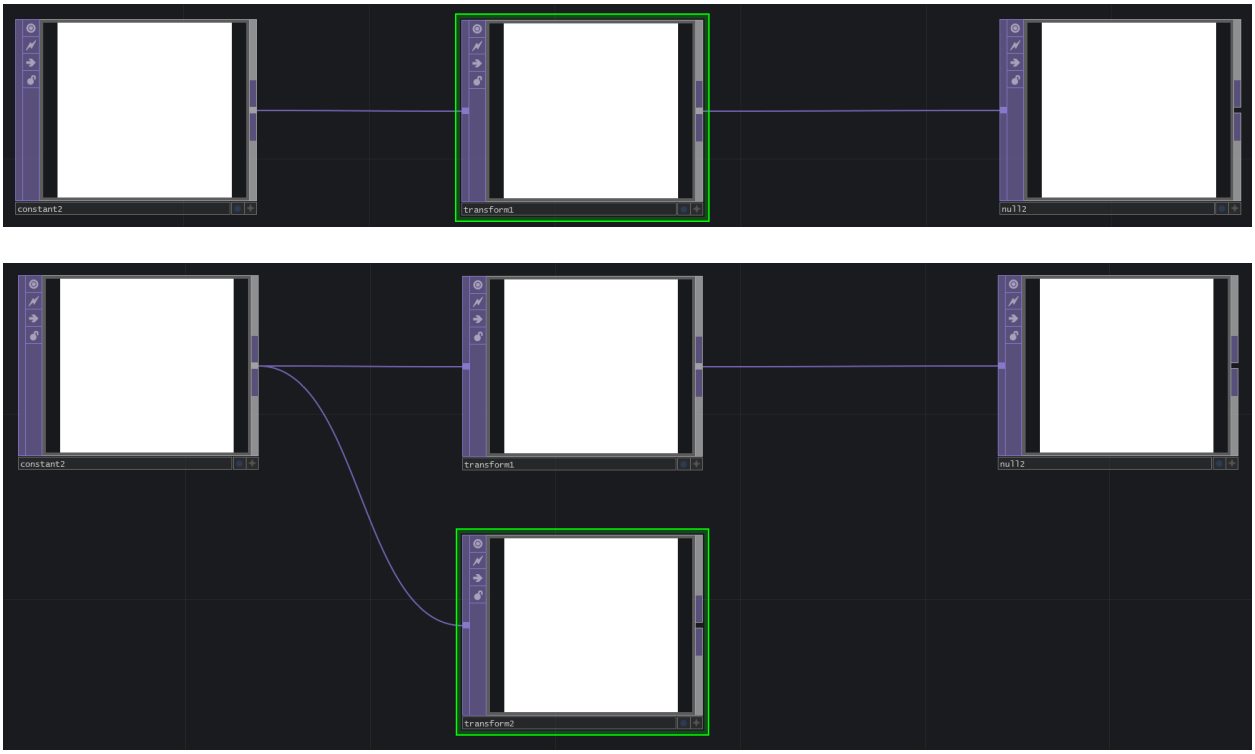
Consideriamo un esempio: abbiamo un TOP “Constant” collegato ad un TOP “Null”, e tra i due deve essere aggiunto un TOP “Transform”. Facendo click con il tasto destro sull’output del TOP “Constant”, oppure sull’input del TOP “Null”, e selezionando il TOP “Transform”, si creerà un TOP “Transform”, che si troverà già collegato tra il TOP “Constant” ed il TOP “Null”.

Il secondo metodo per aggiungere un Operatore ad una catena già esistente consiste nel fare click con il tasto centrale del mouse sull’output di un Operatore. La differenza rispetto al caso precedente è che facendo click con il tasto destro si integra il nuovo Operatore nella catena di Operatori corrente, mentre utilizzare il tasto centrale genera un nuovo ramo in parallelo alla catena di Operatori già esistente.

Risultati simili possono essere ottenuti facendo click con il tasto destro sul filo stesso e selezionando “Insert Operator” o “Add Operator”. “Insert Operator” agisce come il tasto destro sull’output di un Operatore, integrandolo direttamente nella catena di Operatori corrente, mentre “Add Operator” agisce come il click con il tasto centrale, creando un nuovo ramo in parallelo alla catena presente.

Nella figura di seguito è visualizzato un esempio con un TOP “Constant” ed un TOP “Null”. Nella seconda immagine è stato fatto click con il tasto destro del mouse sul filo che collegava i due Operatori, e selezionando “Insert Operator” è stato creato un TOP “Transform”. Nell’ultima figura è stato fatto click con il tasto destro sul filo che connetteva gli Operatori, ed è stato creato un TOP “Transform” scegliendo questa volta “Add Operator”. È da notare come il collegamento avvenga in parallelo al primo TOP “Transform”:





Due tasti sono particolarmente utili quando si lavora con la finestra di creazione degli Operatori: “Control” e “Shift”. Aprite la finestra di creazione degli Operatori, tenete premuto “Control” sulla tastiera e iniziate a selezionare più Operatori di seguito. Ognuno verrà aggiunto al Network al di sotto del precedente. Questa pratica è utile per popolare rapidamente un Network con Operatori diversi.

In modo simile aprite la finestra di creazione degli Operatori, premete e tenete premuto il tasto “Shift”, e poi selezionate più Operatori. La differenza rispetto al caso precedente è il fatto che gli Operatori sono già collegati in serie. Questo tasto può essere utilizzato per creare rapidamente piccole o grandi catene di Operatori già collegati.

Entrambi questi tasti sono piuttosto potenti, ma lo diventano ancora di più quando usati insieme. Consideriamo come esempio un progetto che richieda tre catene di Operatori. La prima consisterà in un TOP “Circle”, collegato ad un TOP “Blur”, collegato ad un TOP “Null”. La seconda comprenderà un TOP “Circle”, un TOP “Edge” ed un TOP “Noise”. L’ultima catena sarà formata da un TOP “Movie In”, un TOP “Blur”, e da un TOP “Null”. Procediamo passo per passo, per mostrare l’uso pratico dei tasti descritti sopra:

1. Aprite la finestra di creazione dei TOP;
2. Premete e tenete premuto il tasto “Shift”;
3. Mentre mantenete premuto “Shift”, fate click sui TOP “Circle”, “Blur” e “Null”. Così creerete la prima catena;
4. Rilasciate il tasto “Shift”;

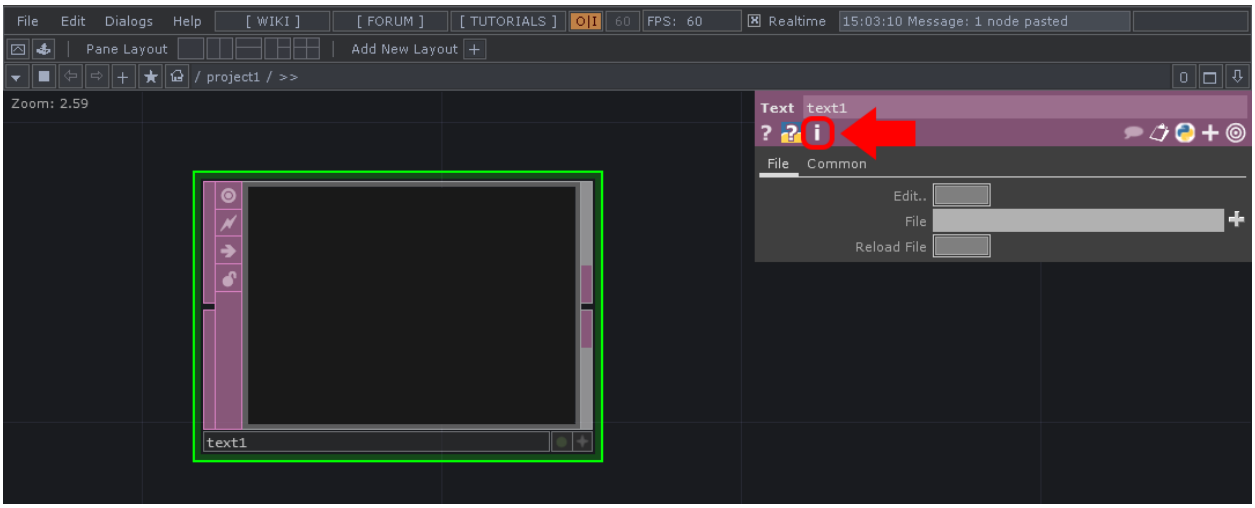
5. Premete e tenete premuto “Control”, in questo modo il prossimo Operatore si posizionerà sotto al primo Operatore;
6. Mantenendo premuto “Control”, fate click sul TOP “Circle”;
7. Rilasciate il tasto “Control”;
8. Premete e tenete premuto il tasto “Shift”;
9. Mentre mantenete premuto “Shift”, fate click sui TOP “Edge”, “Noise” e quindi su “Null”. Così completerete la seconda catena;
10. Rilasciate il tasto “Shift”;
11. Premete e tenete premuto il tasto “Control”;
12. Mantenendo premuto “Control”, fate click sul TOP “Movie In”;
13. Rilasciate il tasto “Control”;
14. Premete e tenete premuto il tasto “Shift”;
15. Fate click sugli operatori rimanenti: i TOP “Blur” e “Null”;
16. Ora che tutti gli Operatori sono stati creati, premete il tasto “Esc” per chiudere la finestra di creazione degli OP.

Dopo aver chiuso la finestra di creazione, tutti gli Operatori richiesti saranno collegati e pronti ad essere utilizzati nel progetto. Questi tasti hanno non solo evitato di dover aprire e chiudere la finestra di creazione degli Operatori per ogni Operatore, ma hanno anche evitato di collegarli manualmente tra loro.

1.3 Navigazione con Mouse e Tastiera

Il mouse gioca un ruolo fondamentale nella programmazione in TouchDesigner, si raccomanda perciò di utilizzarne uno di alta qualità. Il mouse viene utilizzato sia per muoversi all'interno del Network che per lavorare con gli Operatori.

Per navigare nei Network fate click con il pulsante sinistro e trascinate lo sfondo. Per selezionare un Operatore fate click su di esso con il tasto sinistro del mouse. Fate click sinistro e trascinate quell'Operatore per spostarlo. Fate click con il pulsante destro su un Operatore per rivelare un menu con varie voci, che saranno introdotte più avanti. Per selezionare e lavorare con più di un Operatore fate click destro e trascinate il riquadro di selezione intorno agli Operatori desiderati. Fate click con il tasto centrale su un Operatore per avere più informazioni su di esso. Nell'interfaccia utente è presente un pulsante che mostra la stessa finestra di informazioni relative all'Operatore, utile quando il mouse utilizzato non ha il tasto centrale.



Fate click con il pulsante sinistro sulla "i" per avere più informazioni riguardo l'Operatore selezionato.

Ci sono diversi tasti usati per navigare nei progetti di TouchDesigner dalla tastiera. Due di questi tasti sono "U" e "I": premere "U" farà salire di un Network, fuori dal componente corrente, per entrare invece in un Network o in un componente (per esempio in un COMP "Container" o "Base") selezionate il componente e premete "I".

Per centrare lo schermo su tutti gli operatori in un Network usate il tasto "H" sulla tastiera, questo esegue l'azione "Home" nel Network corrente.

1.4 Networks and Percorsi

Tutti i progetti di TouchDesigner si basano sui Network, gruppi di Operatori raggruppati all'interno di "Components", come il COMP "Container", il COMP "Base", il COMP "Geometry", eccetera. I Network possono essere incapsulati all'interno di questi elementi all'infinito. Il livello più alto è detto il livello "root": qui è possibile trovare gli elementi di sistema e dell'interfaccia di TouchDesigner.

Organizzare e racchiudere i Network nei vari "Components" fin dall'inizio di un progetto è un'ottima pratica da allenare per organizzare meglio progetti complessi. Il percorso corrente è sempre visibile nella "Path Bar" in cima al "Network Editor".

Tutti gli Operatori di TouchDesigner hanno un percorso, simile ai percorsi dei file Unix. Ci sono due tipi di percorsi per un Operatore: il "percorso assoluto" ed il "percorso relativo". Il "percorso assoluto" è il percorso per quell'Operatore a partire dal "root" del progetto, indicato con "/". Il "percorso relativo" è invece il percorso dell'Operatore in relazione ad un altro Operatore; questi percorsi vengono indicati partendo dal Network dell'Operatore di riferimento, anziché dal "root" del progetto.

Aprite il file "Paths.toe". Questo esempio mostra i percorsi e le loro differenti nomenclature. TouchDesigner partirà dal "root" del progetto, dove è presente un COMP "Container" chiamato "network1". All'interno di "network1" ci sono due Operatori. Uno, con il nome "rel1", è un DAT "Text" contenente due percorsi. Il primo è un "percorso assoluto": parte dal "root", o dal livello più alto del progetto, e arriva fino all'Operatore. Il secondo percorso è un "percorso relativo", e va dall'Operatore corrente verso "rel2", che è un DAT "Text" posto all'interno del COMP "Container" chiamato "network2". Nel "percorso relativo" l'indicazione parte dal luogo corrente e arriva alla destinazione; per andare da "rel1" a "rel2", il percorso ha bisogno solamente di viaggiare dentro "network2", perciò il "percorso relativo" è "network2/rel2".

Notate che il viewer di "network2" mostra un Operatore al suo interno: questa tecnica sarà discussa in esempi successivi, ma quello che conta adesso è il percorso utilizzato. Nel parametro "Operator Viewer" di "network2" è riportato il percorso "./display", dove "display" è il nome dell'Operatore, e "./" denota un livello all'interno dell'Operatore di riferimento, che in questo caso è "network2".

Dentro a "network2" c'è un DAT "Text" chiamato "display", i cui contenuti sono mostrati nel Network al livello superiore. Gli altri due DAT "Text" hanno altri esempi di percorsi scritti al loro interno: "abs1" è un altro esempio di un "percorso assoluto"; "rel2" contiene invece un esempio di un "percorso relativo" tra se stesso e "abs1"; mostra inoltre un esempio di un "percorso relativo" tra se stesso e "rel1" nel Network soprastante, dove "rel1" è il nome dell'Operatore, e "../" denota il Network ad un livello superiore rispetto al Network corrente. La notazione "../" può essere utilizzata in sequenza per salire fino al livello "root", ma ci sono modi più efficienti di creare questo tipo di percorsi.

1.5 Utilizzare Editor di Testo Esterni

È possibile creare e modificare brevi script in Python direttamente all'interno di TouchDesigner ma, quando gli script aumentano di numero e dimensioni, un editor di testo esterno può far risparmiare molto tempo e fatica.

Editando gli script in questo modo è possibile ottenere numerosi vantaggi, senza voler creare una lista esaustiva, alcuni di questi includono:

1. Numerazione delle righe;
2. Sintassi colour-coded;
3. Funzionalità di ricerca e sostituzione;
4. Auto completamento.

Tutto ciò va a creare un'esperienza più completa e più efficiente quando si lavora in modo estensivo con Python all'interno di TouchDesigner.

1. Selezionate la voce "Preferences" all'interno del menu "Edit";
2. Andate alle preferenze dei "DAT";
3. Fate click sull'icona del file browser relativa alle impostazione del "Text Editor";
4. Assegnate l'eseguibile (.exe) dell'editor esterno selezionandolo e facendo click sul tasto "Open". Il file dell'applicazione è generalmente posto nella cartella "Program Files";
5. Fate click su "Accept" nel menu "Preferences".

Una volta completato il setup, fate click con il tasto destro del mouse su un DAT e scegliete l'opzione "Edit Content": il DAT verrà così aperto nel programma specificato nelle preferenze. Un'opzione separata, indicata come "Table Editor", è disponibile per impostare l'editor esterno da utilizzare per i DAT tabelle.

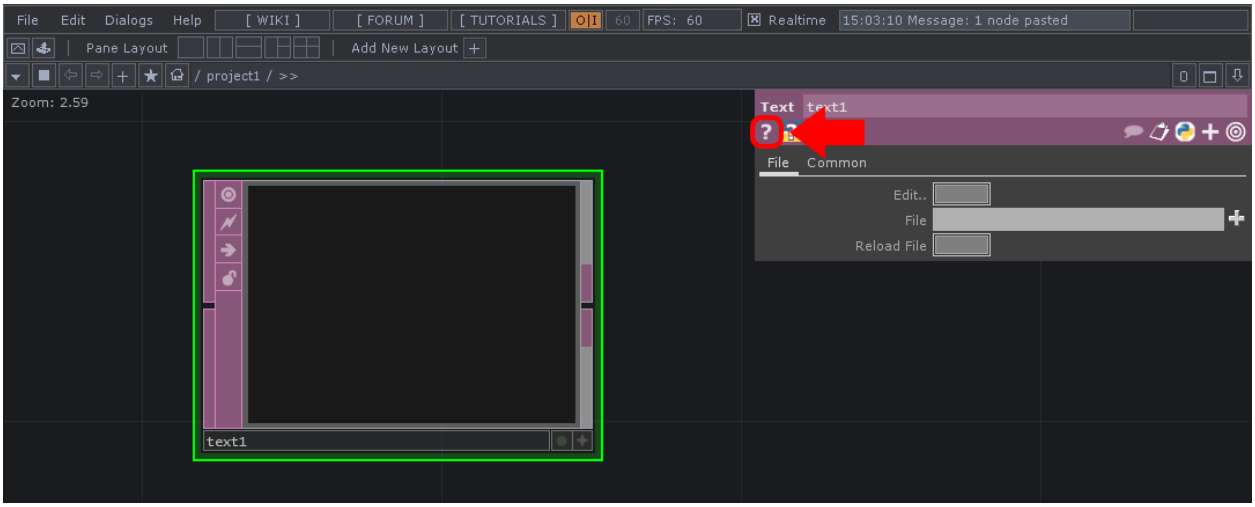
Due editor più che rispettabili utilizzati di frequente nella comunità di TouchDesigner, e che sono crossplatform, sono linkati di seguito:

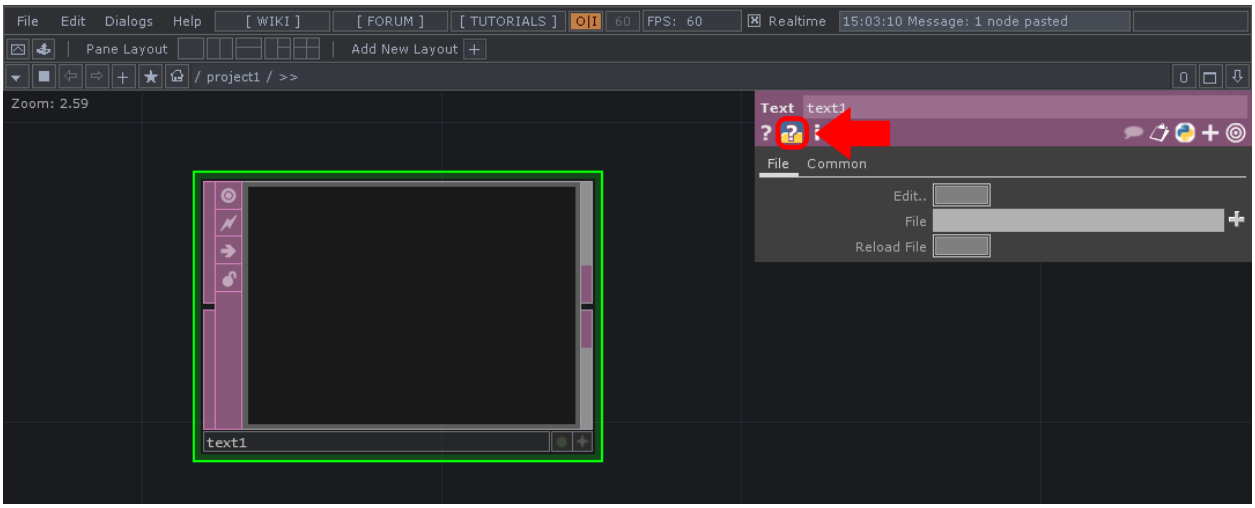
Sublime Text 3 http://www.sublimetext.com/[1] *Notepad++* http://notepad-plus-plus.org/[2]

[1] http://www.sublimetext.com/
[2] http://notepad-plus-plus.org/

1.6 Aiuto

Se doveste mai avere domande riguardo specifici Operatori o processi, fate riferimento alla Wiki ufficiale di Derivative. Ogni Operatore ha due scorciatoie che aprono riferimenti alla Wiki in una nuova finestra del browser. Questi due pulsanti sono posizionati nella finestra dei parametri e sono entrambi rappresentati da punti di domanda: uno riguarda lâ€™Operatore ed il suo utilizzo specifico, mentre lâ€™altro, con il punto di domanda sovrapposto al logo di Python, si riferisce allo scripting in Python di quellâ€™Operatore.





2 Interfaccia Utente

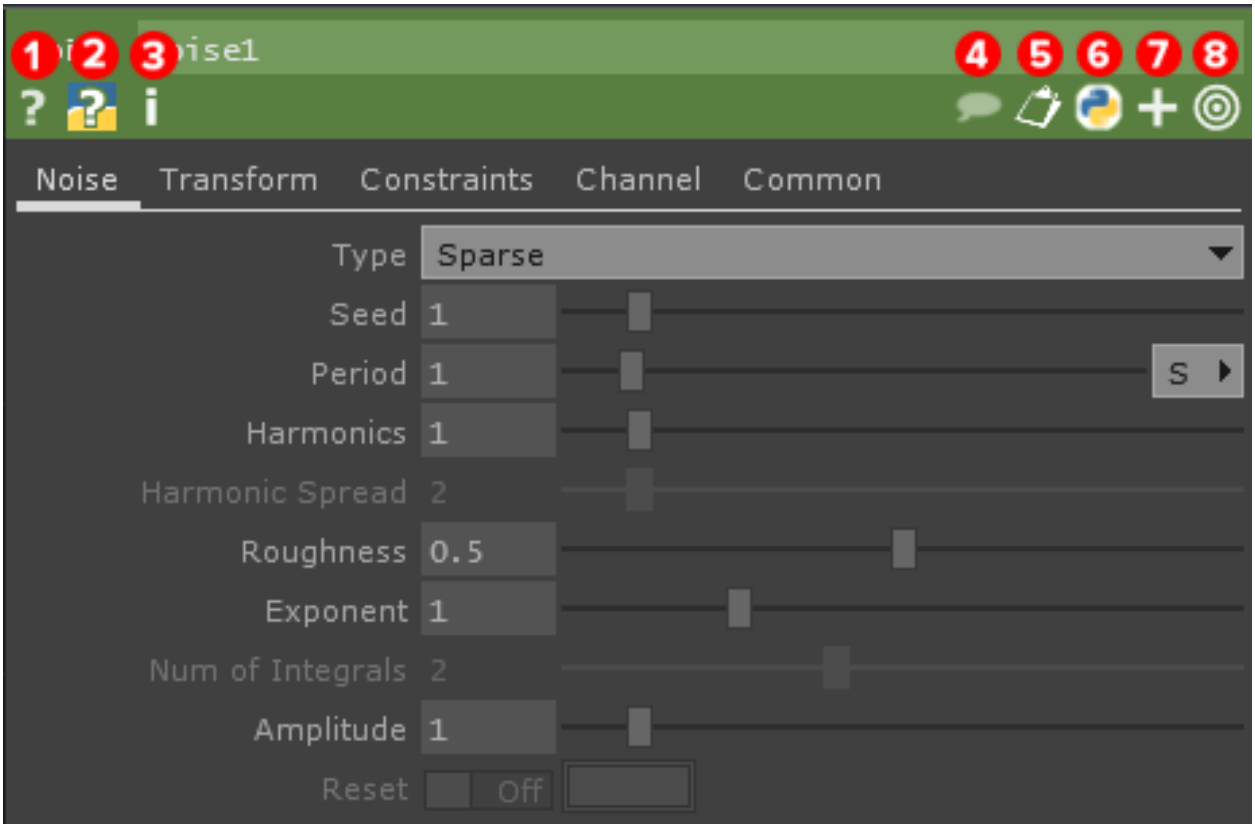
2.1 Finestra dei Parametri

Tutti i valori e le proprietà di un Operatore possono essere controllati dalla finestra dei parametri.

Ci sono due modi di accedere alla finestra dei parametri: il primo è utilizzando il tasto “p”. Questo aprirà un riquadro agganciato all’angolo superiore destro del pannello. Questa finestra mostrerà i parametri di qualsiasi Operatore selezionato.

Il secondo modo di accedere alla finestra dei parametri è facendo click con il tasto destro del mouse su un Operatore e selezionando “Parameters...”. Così si aprirà una finestra flottante relativa a tale Operatore. Questo metodo si differenzia dal primo per il fatto che i parametri non cambieranno se si dovesse selezionare un altro Operatore, caratteristica molto utile per essere in grado di gestire parametri di più Operatori contemporaneamente.

Ogni Operatore conta un diverso insieme di proprietà, ma tutte le finestre dei parametri hanno le stesse opzioni; di seguito è uno schema che le evidenzia:



Da sinistra a destra, le opzioni sono come segue:

1. Aiuto dell'Operatore: apre in una nuova finestra del browser la pagina di aiuto della Wiki relativa all'Operatore;
2. Aiuto di Python dell'Operatore: apre in una nuova finestra del browser la pagina di aiuto della Wiki relativa allo scripting in Python dell'Operatore;
3. Menu di informazioni dell'Operatore: mostra informazioni riguardo il funzionamento dell'Operatore, in modo simile a quando si fa click con il tasto centrale del mouse sull'Operatore;
4. Commenti: mostra e consente di modificare i commenti dell'Operatore;
5. Parametri copiati: mostra i parametri copiati utilizzando il menu con il tasto destro del mouse;
6. Linguaggio: permette di selezionare se l'Operatore userà Python o tscript come linguaggio di scripting;
7. Espandi/Nascondi Parametri: espande o nasconde tutti i parametri dell'Operatore;
8. Parametri non-default: mostra solo i parametri che sono stati modificati rispetto ai loro valori di default.

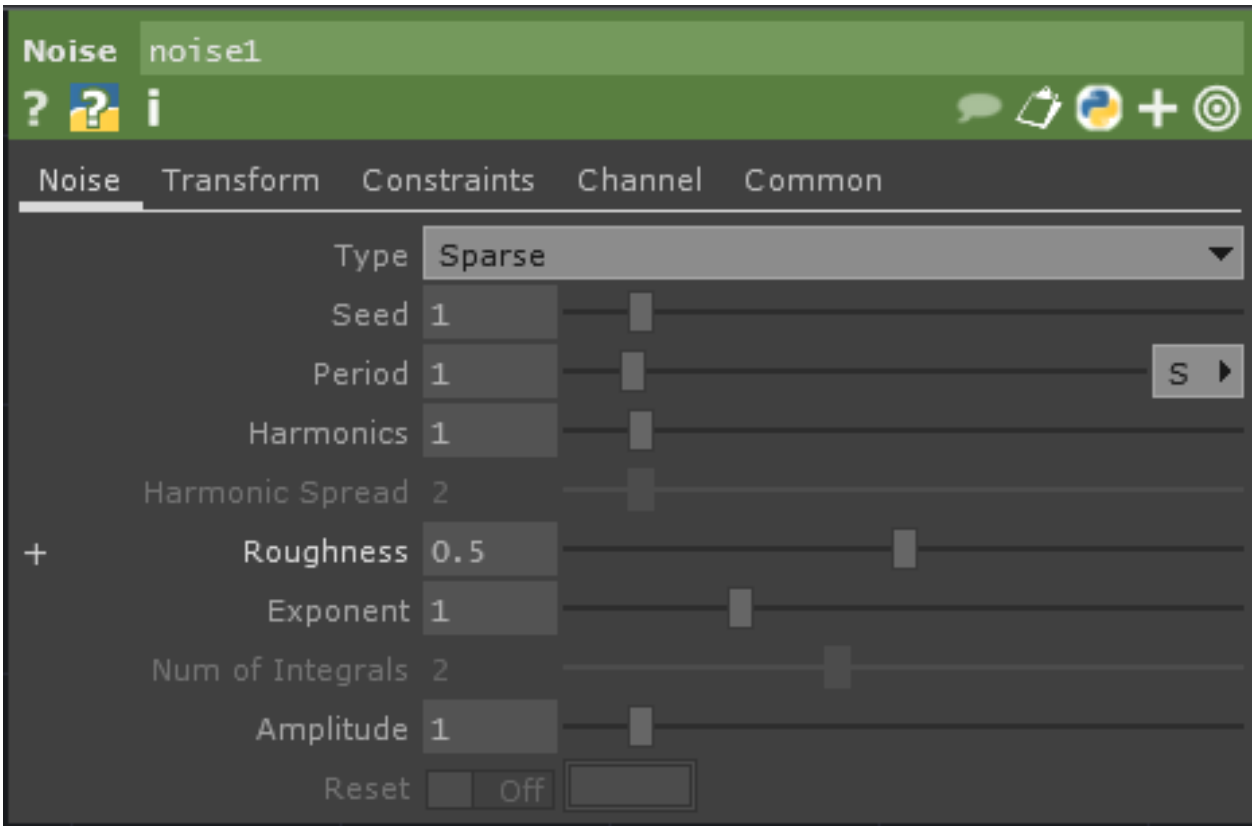
2.2 Parametri

I parametri possono essere inseriti in molti modi. A seconda della situazione, alcuni parametri posso richiedere un valore statico, mentre altri possono aver bisogno di essere comandati da valori ed input esterni. Ogni parametro ha tre modalità, ed ogni modalità è piuttosto differente, definendo in modo diverso il comportamento del parametro. Le tre modalità sono:

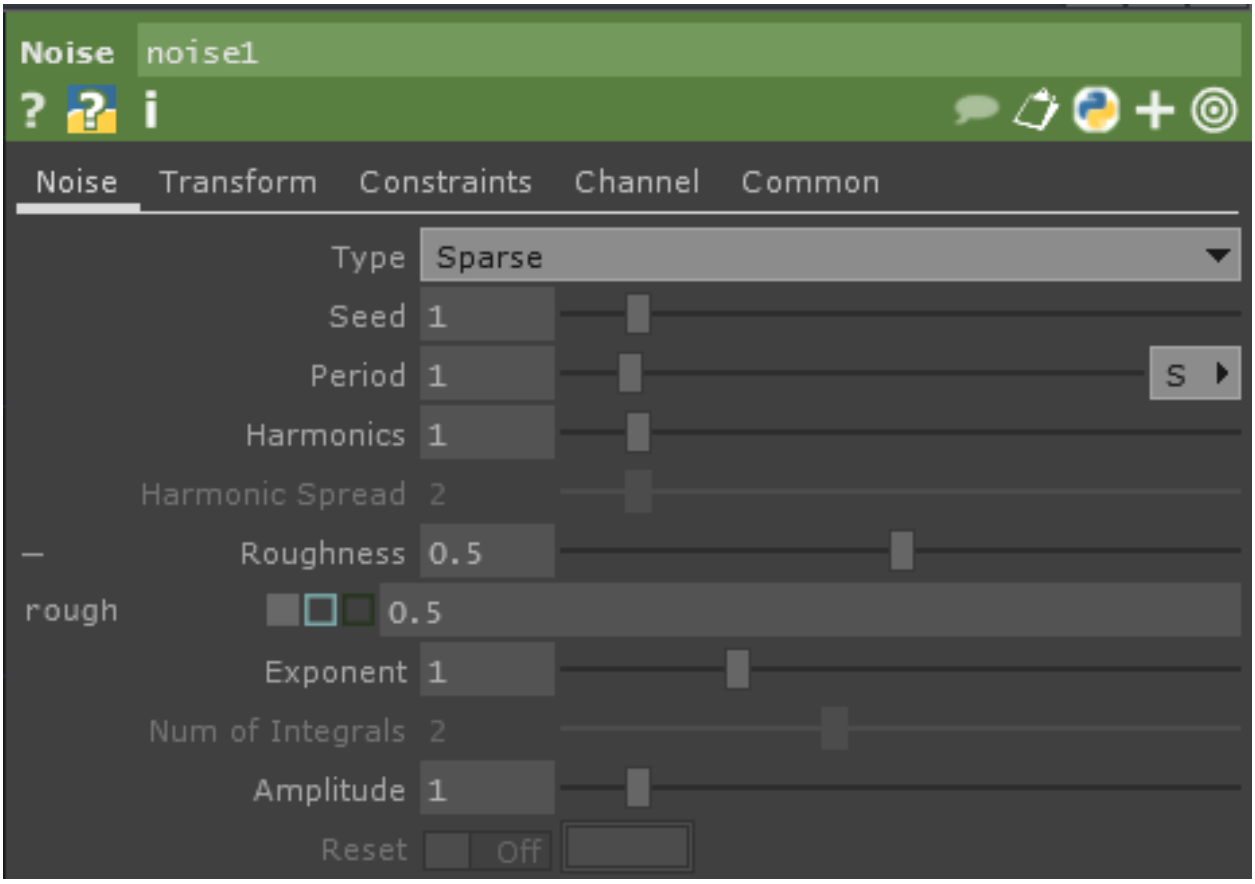
1. Costante;
2. Espressione;
3. Esportazione.

La modalità costante è quella di default per la maggior parte dei parametri, ed è rappresentata nel campo del valore da uno schema di colori grigio. La modalità espressione è utilizzata per Python, tscript, o per operazioni matematiche e script, ed è rappresentata da uno schema colori grigio scuro e blu chiaro. La modalità esportazione infine è utilizzata per fare riferimento diretto ai canali dei CHOP, ed è rappresentata da uno schema di colori verde chiaro.

Ciascuno dei parametri di un Operatore può essere impostato indipendentemente in una delle tre modalità. Per cambiare la modalità, occorre passare il mouse sopra il nome del parametro; un segno "+" apparirà vicino al nome del parametro, come mostrato nella figura di seguito:



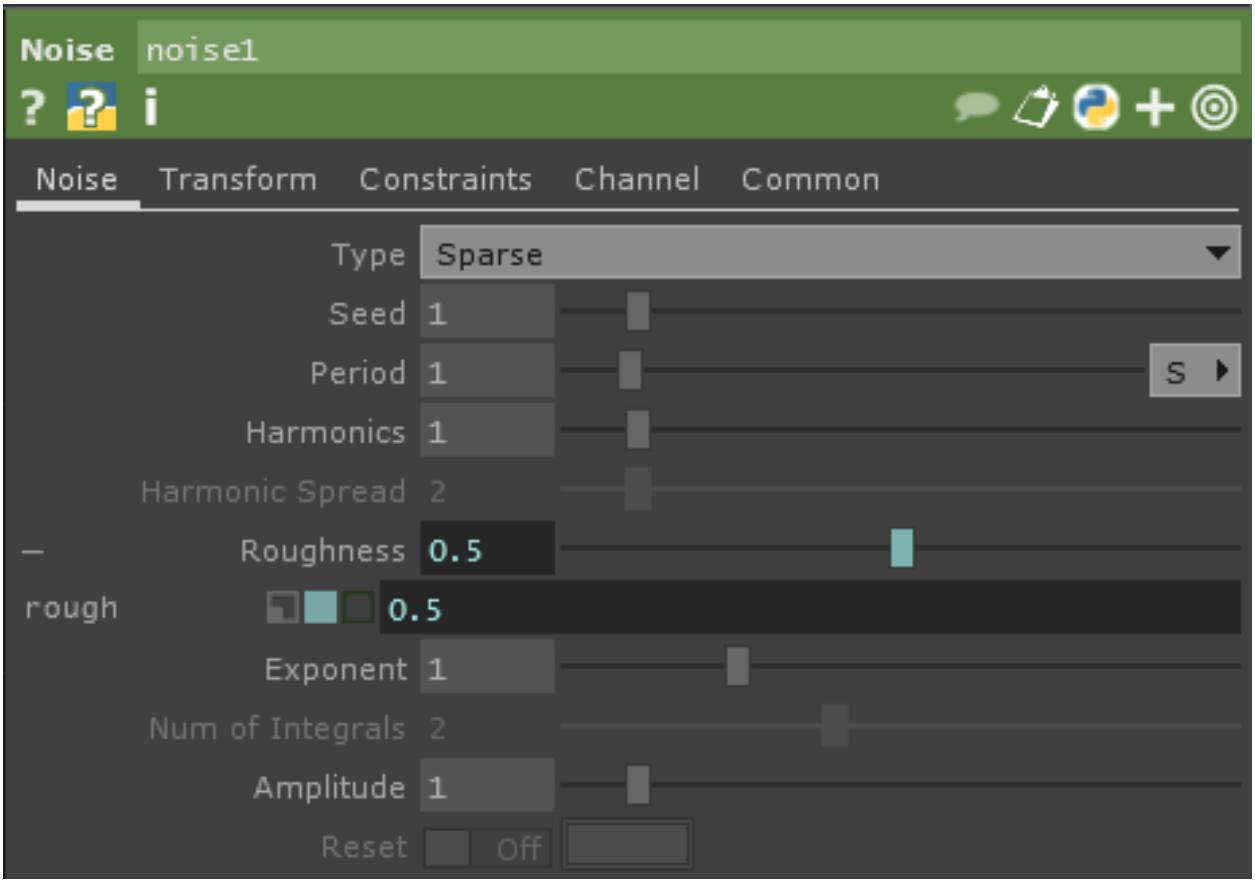
Una volta che il mouse è sopra al nome del parametro, è possibile fare click per espanderlo, mostrando più informazioni ed opzioni, in modo simile alla figura seguente:



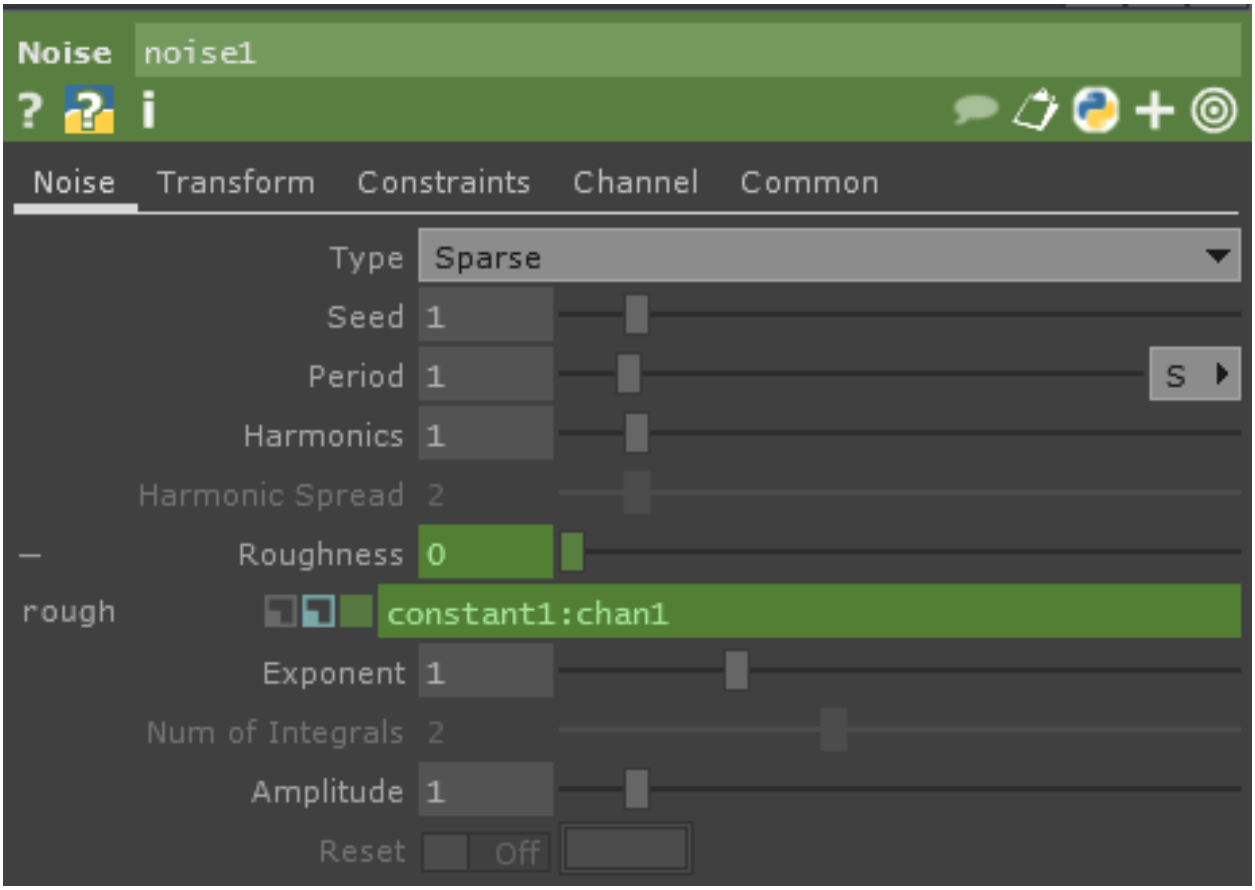
Una volta espanso un parametro sono presenti tre elementi principali: il primo sulla sinistra è il nome per lo scripting del parametro. Questo nome è necessario ogni qual volta si fa riferimento a quel parametro in uno dei linguaggi di scripting di TouchDesigner. Nella figura precedente, il nome di scripting del parametro "Roughness" del CHOP "Noise" è "rough". Continuando con l'esempio di prima, lo script in Python per impostare il valore "Roughness" del CHOP "Noise" ad "1" sarebbe:

```
1  op('noise1').par.rough = 1
```

Il secondo elemento sono i tre quadrati colorati. Questi quadrati rappresentano le diverse modalità dei parametri, come elencate prima. In modalità costante i parametri di un Operatore sono rappresentati da un quadrato riempito di grigio. Questo parametro può essere cambiato in modalità espressione cliccando sul quadrato dal contorno azzurro; una volta selezionato, il quadrato azzurro si riempirà, ed il campo per il valore sulla destra risulterà anch'esso colorato per rispecchiare la modalità del parametro.



Per cambiare la modalità del parametro in esportazione, occorre trascinare e rilasciare il canale di un CHOP sul parametro, che prenderà quindi lo schema di colori della modalità esportazione, ed il quadrato verde si riempirà.



Il terzo elemento del parametro espanso è il campo dei valori, che mostra informazioni diverse a seconda della modalità del parametro. In modalità costante il campo dei valori mostra il valore corrente, e può essere modificato facendo click e digitando nella casella. In modalità espressione il campo dei valori mostra lo script di Python, o di tscript, che viene eseguito. L'espressione può essere modificata facendo click e scrivendo nel campo dei valori. Infine in modalità esportazione il campo dei valori mostra due informazioni separate da ":". Il testo prima dei due punti mostra il percorso del CHOP che sta esportando il proprio valore verso questo parametro; il testo dopo i due punti è il nome del canale che viene esportato da quel CHOP. Poiché questi valori sono imposti da un altro Operatore, il campo dei valori non può essere modificato fintanto che il parametro è in modalità esportazione.

2.3 Controlli di Trasporto

La barra di trasporto funziona in modo simile a quella di molte altre applicazioni. Affrontiamola rapidamente, mostrando da sinistra a destra la funzione dei vari pulsanti:

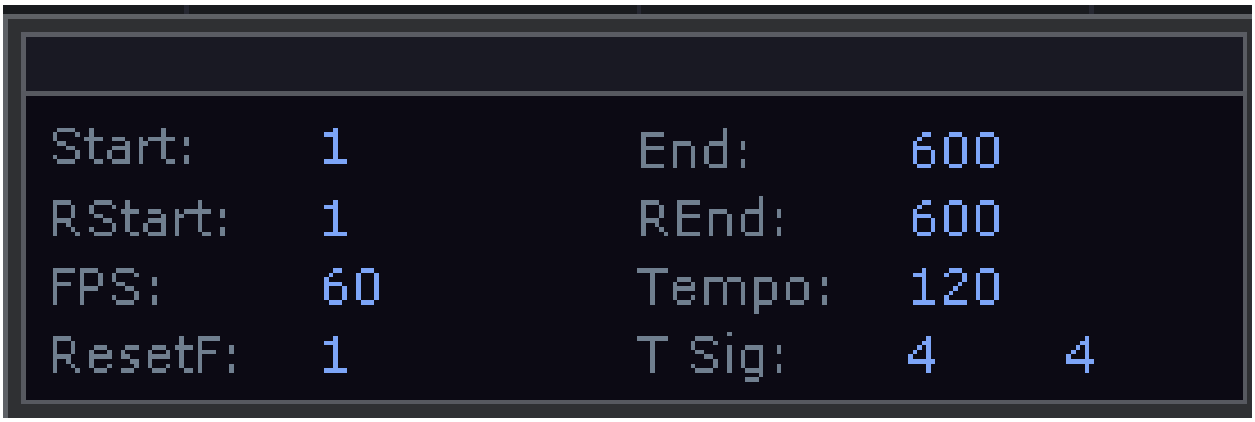
1. Resetta la timeline al primo fotogramma;
2. Mette in pause la timeline;
3. Fa partire in reverse la timeline;
4. Fa partire la timeline;
5. Torna indietro di un frame;
6. Va avanti di un frame;
7. Permette di ciclare continuamente la timeline impostando il “Range Limit” su “Loop”;
8. Permette di far scorrere la timeline e di mantenere l’ultimo frame impostando il “Range Limit” su “Once”.

Le funzioni più utilizzate sono “Play” e “Pausa”, che possono essere raggiunte facilmente premendo la barra spaziatrice.

2.4 Impostazioni della Timeline

Le impostazioni della timeline non avranno normalmente bisogno di essere modificate, a meno di contenuti o animazioni legati ad essa. Le impostazioni della timeline si trovano in fondo a sinistra nella finestra. La cosa fondamentale da sapere riguardo quest'area è che gli FPS ed il Tempo del progetto possono essere cambiati da qui. Gli "FPS" del progetto determinano la frequenza a cui il progetto renderizza i fotogrammi. Di default è impostato a 60 FPS, che significa che TouchDesigner proverà a renderizzare 60 frame ogni secondo. Il "Tempo" imposterà i BPM (battiti per minuto) del progetto, per l'utilizzo da parte del CHOP "Beat".

Le impostazioni della timeline vengono utilizzate prevalentemente in situazioni in cui le animazioni ed i vari media hanno bisogno di essere bloccati ad una timeline stabile. I controlli dei frame includono "Start" e "End", che controllano rispettivamente il fotogramma di partenza e quello finale, così come "RStart" e "REnd", che controllano l'inizio e la fine del loop nella timeline. Con queste impostazioni è possibile creare un'animazione che si estende sull'intera timeline, che può essere lunga 4000 frame, lasciando comunque la possibilità di ciclare una piccola sezione per lavorarci.

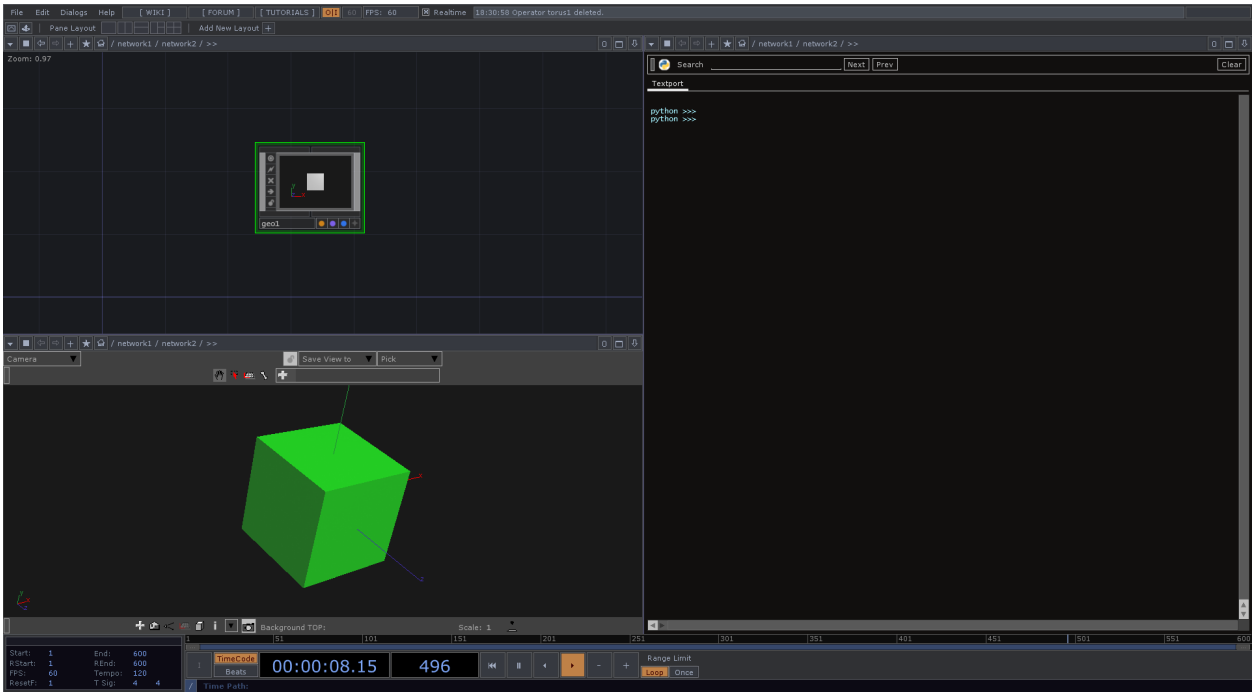


2.5 Pannelli

Utilizzare regolarmente i pannelli può far risparmiare moltissimo tempo quando ci si sposta avanti e indietro fra i Network. Dover passare attraverso tre Network per cambiare un parametro solo per dover poi ritornare indietro a controllare il risultato è una perdita di tempo. I pannelli prendono la finestra corrente e la dividono orizzontalmente o verticalmente quante volte si desidera. Ogni layout dei pannelli può essere salvato per venire riutilizzato successivamente.

La figura precedente evidenzia i preset dei pannelli disponibili di default. Questi preset offrono un accesso rapido ad alcune configurazioni standard dei pannelli, incluse la divisione destra e sinistra, sopra e sotto, un'impostazione a tre pannelli ed una a quattro pannelli. Per salvare un nuovo preset dei pannelli è sufficiente fare click sul pulsante "Add New Layout +", inserire un nome, e selezionare "Ok". Salvare un layout salva non solo la dimensione e la posizione dei pannelli, ma anche la tipologia di ogni pannello.

I pannelli sono in grado di mostrare tipologie uniche di contenuto, siano essi menu, Network o visualizzatori. Essere in grado di mescolare e affiancare combinazioni diverse di visualizzatori e di editor permette moltissima flessibilità. Nella figura di seguente il pannello in alto a sinistra è un editor del Network, nel lato destro c'è un Textport e in basso a sinistra c'è un visualizzatore di geometrie. È importante ricordare che salvare questo layout salverebbe non solo la disposizione dei pannelli, ma anche le tipologie dei vari pannelli. Questo diventa incredibilmente utile quando si lavora su un progetto con molti elementi differenti, dove, a lungo andare, saltare direttamente ad un setup simile, rispetto che ad un semplice editor del Network, può far risparmiare parecchio tempo.



Le scorciatoie da tastiera per lavorare con i pannelli sono le seguenti:

1. Alt + [: divide verticalmente il pannello in cui si trova il puntatore del mouse;
2. Alt +] : divide orizzontalmente il pannello in cui si trova il puntatore del mouse;
3. Alt + Z : chiude il pannello in cui si trova il puntatore del mouse.

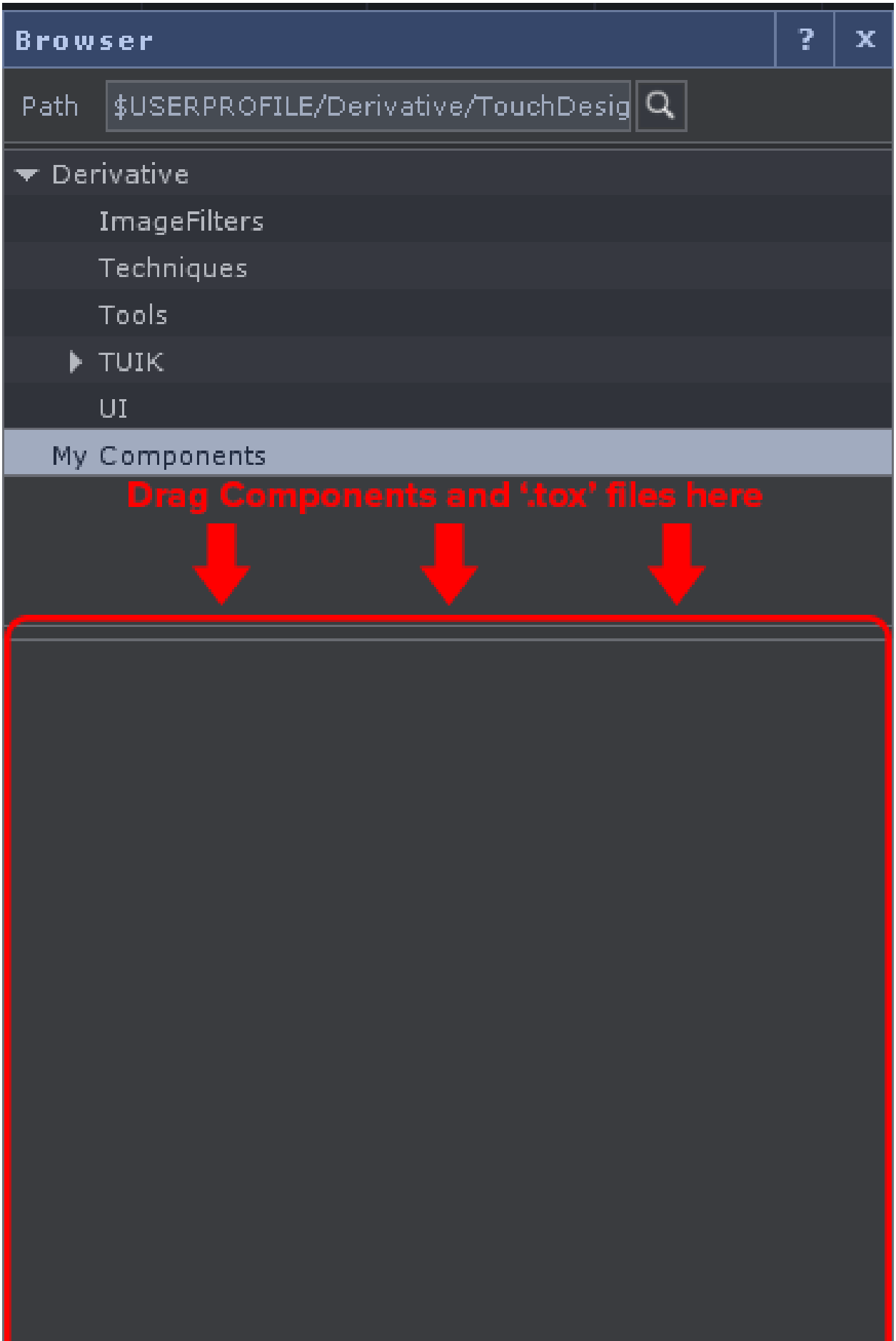
2.6 Browser della Palette

Il browser della palette può essere visto come una libreria di componenti contenente file ".tox" (o file TouchDesigner Component). Ognuno di questi file comprende un singolo Operatore COMP, che può a sua volta contenere un Network di altri Operatori. Ciò significa che una serie di Operatori usati frequentemente, componenti dell'interfaccia utente, script Python, o altri, possono essere racchiusi all'interno di un singolo Operatore COMP, salvato poi come file ".tox" e accessibile rapidamente in ogni momento.

Aprite il browser della palette, e osservate il gran numero di file ".tox" già disponibili. I progetti vuoti iniziano con la palette aperta di default, agganciata nel lato sinistro della finestra. Per aprire il browser della palette come finestra flottante, usate la combinazione di tasti "Alt+L" sulla tastiera. Proviamo uno dei componenti pre-costruiti.

Nella sezione "Derivative" navigate su "Tools", e quindi trascinate e rilasciate il componente "Blend" in un nuovo progetto. Guardando l'interfaccia utente del componente "Blend" è evidente che al suo interno avvengano un po' di cose diverse. Prima di tuffarci nella sua analisi prendetevi un momento per collegare due input e testare il componente "Blend". Attivate il visualizzatore, cliccate sul pulsante sottostante all'immagine per scegliere un metodo di fusione e trascinate quindi la maniglia semi-trasparente lungo l'immagine per miscelare i due input. Questo strumento è molto utile, e per averlo è stato sufficiente trascinarlo e rilasciarlo dal browser della palette!

Uno degli obiettivi di questo libro è creare alcuni strumenti che possano essere aggiunti al browser della palette, così da poterli poi utilizzare regolarmente. Ci sono due metodi di aggiungere un componente al browser della palette: il primo metodo consiste nel trascinarlo e rilasciarlo. Per fare ciò, selezionate "My Components" dalla parte superiore del browser, quindi trascinate qualsiasi componente dal Network e rilasciatelo nella parte inferiore del browser della palette; verrà così aggiunto alla categoria "My Components". Il secondo metodo per aggiungere un componente è trascinare un file ".tox" già salvato da Windows Explorer, e rilasciarlo nella stessa parte menzionata prima. La figura mostra esattamente dove bisogna rilasciare i componenti:


Browser
?
x
Path
$USERPROFILE/Derivative/TouchDesig
Derivative
ImageFilters
Techniques
Tools
TUIK
UI
My Components
Drag Components and '.tox' files here

2.7 Menu di Ricerca

Il menu di ricerca è incredibilmente utile quando i progetti diventano più complicati e popolati di script Python e di Network innestati uno nell'altro: può trovare una moltitudine di elementi in molti luoghi diversi, e le ricerche possono essere tanto ampliate, o ristrette, quanto necessario. È accessibile nel menu "Edit" in cima allo schermo, o premendo "F3" sulla tastiera.

La ricerca di base può trovare Operatori, ma può anche cercare all'interno degli script Python. Spesso infatti Python è utilizzato per cambiare i valori dei parametri di un Operatore, e a volte, nella lunghezza degli script, è facile perdere di vista una specifica linea di codice. Una veloce ricerca del codice riportato sotto restituirà una lista di ogni singola riga che coinvolge il cambiamento di parametri degli Operatori con "transform" nel loro nome:

```
1  op('transform').par
```

Scorrere attraverso i risultati è molto più facile che cercare manualmente all'interno di Network pieni di codice.

La ricerca avanzata può cercare filtrando ogni combinazione di nome, tipologia di Operatore, commenti e altro. Quando si riaprono progetti passati spesso è necessario del tempo per ambientarsi e ritrovare i funzionamenti interni di sistemi dalle logiche complesse. Questo è un caso in cui cercare gli operatori attraverso nomi generici può far risparmiare un sacco di tempo. Per esempio, in un sistema estremamente nidificato, da qualche parte nelle sue profondità può trovarsi un TOP "Movie In" che ha la parola "movie" nel suo nome. Questi piccoli frammenti di informazioni riguardo il tipo di Operatore ed il suo nome parziale possono essere utilizzati per creare una ricerca abbastanza precisa.

Quando una ricerca restituisce i suoi risultati, ogni esito può essere selezionato per aprire una nuova finestra flottante del Network: qui i risultati possono essere visualizzati rapidamente, ed è possibile apportare piccoli cambiamenti senza influenzare l'area di lavoro corrente.

2.8 Contrassegno Realtime

Il contrassegno realtime cambia il comportamento di TouchDesigner in modo significativo. Quando è attivo (lo è di default), TouchDesigner darà sempre priorità al mondo reale, quindi se per esempio un filmato è lungo 30 secondi, TouchDesigner cercherà di riprodurlo nell'arco di 30 secondi, non importa cosa succede: se ciò comporta perdite di fotogrammi, TouchDesigner cercherà comunque di rispettare la durata. Questo è il metodo utilizzato per la maggior parte delle installazioni realtime e per i lavori di performance.

Quando il contrassegno realtime è spento, TouchDesigner darà priorità a renderizzare i frame piuttosto che al tempo del mondo reale. Nell'esempio menzionato prima, se il contrassegno realtime non è selezionato, TouchDesigner impiegherà tutto il tempo necessario a processare e renderizzare ogni frame, trascurando il tempo del mondo reale per mostrare ogni fotogramma. Questo metodo è utile quando si esportano animazioni complesse o rendering 3D. Immaginate questa modalità come simile alla versione realtime di un render compiuto da Adobe After Effects.

2.9 Scorciatoie Utili

Di seguito è riportato un elenco di alcune utili scorciatoie da tastiera:

Quando il mouse è in un Network:

- 'P' - Apre e chiude la finestra dei parametri dell'Operatore selezionato;
- 'O' - Apre e chiude un'anteprima del network nell'angolo in basso a sinistra del pannello;
- 'C' - Apre e chiude la Palette dei colori. Così è possibile aggiungere un bordo colorato all'Operatore selezionato per un'identificazione più facile;

Quando è selezionato un Operatore (o più di uno):

- 'A' - Permette di interagire con il visualizzatore dell'Operatore;
- 'B' - Disattiva e riaccende l'Operatore selezionato;
- 'H' - Compie nel Network l'azione "Home All", che equivale a mostrare sullo schermo tutti gli Operatori di un Network;
- 'Shift + H' - Compie l'azione "Home Selected", che equivale a mostrare sullo schermo tutti gli Operatori selezionati;
- 'R' - Accende e spegne il contrassegno di render dell'Operatore (se ne ha uno);
- 'D' - Accende e spegne il contrassegno display dell'Operatore (se ne ha uno);
- 'Control + C' - Copia gli Operatori selezionati;
- 'Control + V' - Incolla gli Operatori copiati;
- 'Control + Shift + V' - Incolla sulla posizione del mouse gli Operatori copiati.

3 TOP

3.1 Introduzione

Gli Operatori Texture, o TOP, sono un aspetto fondamentale di quasi ogni progetto. Sono gli Operatori che si occupano delle texture 2D, e gestiscono moltissimi aspetti: la riproduzione dei filmati, il rendering delle geometrie 3D, il compositing, coordinano gli ingressi e le uscite hardware e sono utilizzati per processare tutto ciÃ² che va inviato a monitor, proiettori o display LED.

3.2 TOP Movie In

Il TOP “Movie In” è uno dei TOP più utilizzati. La sua funzione è quella di caricare gli asset in TouchDesigner.: è in grado di caricare molti tipi diversi di media, partendo dalle immagini fisse fino ad una varietà di codec per i filmati. Di seguito è riportata una piccola lista di formati di file comunemente utilizzati con il TOP “Movie In”:

- .mov
- .mp4
- .avi
- .tiff
- .jpeg
- .png

Nella Wiki di TouchDesigner, alla pagina “File Types”, sono elencati molti altri formati di file supportati.

Il TOP “Movie In” integra alcune ottime funzioni che possono significativamente ridurre i problemi derivanti dalla necessità di caricare e riprodurre file con frequenze di aggiornamento differenti. La caratteristica principale è che il TOP “Movie In” cerca sempre di riprodurre i media rimanendo fedele alla loro durata: per esempio, se il progetto è impostato a 60 FPS, e un asset registrato a 30 FPS è lungo 10 secondi, questo verrà riprodotto nel corso di 10 secondi, indipendentemente dalla differenza di frequenza di aggiornamento tra il progetto ed il file. Vale anche il contrario: se un secondo asset, che dura 10 secondi a 60 FPS, è riprodotto in una timeline a 30 FPS, esso verrà riprodotto nel corso di 10 secondi. In entrambi i casi i frame sono duplicati o scartati per rimanere fedeli alla durata dei media nel mondo reale. Ciò rende una buona idea, in alcune situazioni, interpolare i frame.

3.3 Precaricare i Filmati

Quando si creano applicazioni realtime, la perdita continua di fotogrammi può ridurre notevolmente l'impatto della presentazione; diagnosticare i problemi di performance verrà discusso in un capitolo più avanzato, ma è già possibile assumere molte misure cautelari. La semplice procedura di caricamento e rimozione dalla memoria è spesso trascurata dei nuovi utenti, perché i metodi più semplici coinvolgono lo scripting.

Aprite l'esempio "Preloading.toe". Questo esempio contiene tre pulsanti. Il primo, chiamato "Pre-load", utilizza la seguente funzione di Python per caricare il numero di frame impostato nel parametro "Pre-Read" all'interno dei parametri "Tune" dell'Operatore "moviein1":

```
op('moviein1').preload()
```

Il pulsante "Play" fa invece partire la riproduzione del TOP "Movie In", mentre il tasto "Unload" arresta la riproduzione del file "moviein1", e rimuove quindi dalla memoria il filmato, liberando le risorse di sistema utilizzate. È possibile fare ciò sfruttando il seguente script Python:

```
op('play').click(0)
```

```
op('moviein1').unload()
```

È un'ottima abitudine caricare i filmati prima di eseguirli, per evitare il rischio di perdere fotogrammi durante la riproduzione.

3.4 TOP Null e TOP Select

A differenza di TOP più esigenti, come il TOP “Blur”, alcuni TOP sono “gratuiti” e dovrebbero essere utilizzati con abbondanza! Due esempi specifici sono i TOP “Null” e i TOP “Select”: questi due Operatori, nonostante non alterino alcun pixel, sono incredibilmente utili per creare workflow più efficienti.

La differenza tra un Network illeggibile, con collegamenti che si sovrappongono e sparsi ovunque, ed un Network facile da seguire, risiede nel semplice utilizzo di alcuni TOP “Null” e “Select”, posizionati debitamente. Aprite gli esempi “Null_1.toe” e “Null_2.toe”. Il primo file è un insieme disordinato di TOP compositati assieme: in questo file c’è poca cura per il layout del Network, e i fili di collegamento sono sovrapposti ad altri Operatori e ad altri fili, rendendo difficile seguire una particolare sequenza di Operatori. Invece, all’interno del progetto “Null_2.toe”, una colonna di TOP “Null” è impiegata per raccogliere tutti i segnali prima di compositarli. Questa colonna di TOP “Null” può essere utile come punto di controllo, e anche ad una rapida occhiata rende molto più facile seguire le diverse serie di operazioni.

Lo stesso esempio può essere fatto per i TOP “Select”. Quando si lavora con Network innestati uno dentro l’altro, usare il TOP “Out” per costruire connessioni tra i COMP “Container” può portare alla stessa situazione di prima, dove i Network diventano illeggibili piuttosto rapidamente. I TOP “Select” possono fare riferimento ad altri TOP in modo veloce e pulito. Aprite l’esempio “Select_1.toe”. Questo progetto mostra come l’utilizzo di TOP “In” e “Out” possa portare ad una gran confusione, pur replicando il filmato solo 12 volte! Cosa succederebbe se ne servissero 100? Ecco dove torna utile il TOP “Select”.

Aprite l’esempio “Select_2.toe”. Questo progetto aumenta esponenzialmente la quantità di componenti replicate, pur mantenendo una maggiore leggibilità. Ancora più interessante è il sistema di selezione dinamica creato utilizzando i TOP “Select”: è un metodo molto più efficiente della selezione manuale usata prima, e consente ad uno script Python inserito nel parametro “Select” del TOP “Select” di fare riferimento in modo automatico ai TOP replicati dal Network al livello superiore, basandosi sui numeri nel loro nome. Per portare questo concetto un passo più in là, un DAT “Select” è utilizzato per pilotare un COMP “Replicator”: questo crea un nuovo TOP “Select” e lo collega automaticamente al TOP “Composite” ogni volta che, con il primo COMP “Replicator”, viene aggiunto un elemento. Non preoccupatevi se questo esempio sembra complicato, i replicatori e lo scripting verranno affrontati in esempi più avanzati; per adesso la cosa importante da notare è che, utilizzando TOP “Select” e scripting molto semplici, questo componente è al sicuro da futuri aggiornamenti, e non richiederà molta manutenzione. Quando sarà necessario replicare più oggetti, sarà sufficiente aggiungere una riga ad una tabella.

3.5 Codec

La riproduzione di filmati è un processo dispendioso. Sarebbe saggio spendere del tempo facendo prove ed esperimenti con codec differenti, per vedere, in termini di qualità visiva e performance, quale sia il più adatto ad un determinato progetto.

Prima di addentrarci nei codec specifici è importante conoscere la differenza tra un codec e un container: il termine codec si è infatti affermato come nome generico per i formati dei file audio-video, il che può confondere i principianti, dato che molti container possono contenere differenti codec.

Codec sta per compressione-decompressione. Il codec ha due compiti principali: il primo è comprimere i dati video per l'archiviazione ed il trasporto, il secondo è decomprimere i dati video per la riproduzione. Visti questi differenti compiti, ogni codec è creato per scopi differenti: alcuni danno priorità alla compressione, per avere file leggeri e trasportabili, mentre altri preferiscono dare priorità alla qualità, per la conservazione a lungo termine del contenuto. Progetti differenti hanno differenti necessità, a volte l'obiettivo è riprodurre un singolo contenuto con la qualità più alta possibile, mentre altre volte è necessario sacrificare la qualità per essere in grado di riprodurre più file video contemporaneamente. Trovare il codec giusto per ogni progetto può richiedere molti test e analisi, ma è tempo ben speso.

Un container fa esattamente ciò che indica il suo nome: contiene il video compresso, l'audio e tutti i metadata di cui un player video ha bisogno per decomprimere adeguatamente e riprodurre il contenuto. Ci sono molti diversi tipi di container, ma hanno molto meno impatto sul flusso di lavoro e di programmazione rispetto ai codec.

Le cose possono complicarsi quando vengono utilizzate diverse combinazioni di container e codec: immaginate un file chiamato "test_movie.mov". In un primo esempio questo file potrebbe essere un video compresso con un codec Animation all'interno di un container QuickTime ".mov". Quello che è interessante, e quello che spesso confonde molti principianti, è che in un secondo esempio questo stesso file potrebbe essere un video compresso in H.264 all'interno di un container QuickTime. Per aumentare il disordine, lo stesso file H.264 potrebbe trovarsi all'interno di un container ".mp4", o MPEG-4 Part 14.

Confusione a parte, alcune popolari scelte di codec al momento sono la famiglia HAP, H.264, Animation e Cineform. Ogni codec ha il suo insieme di vantaggi e svantaggi. Di seguito è riportato per ognuno un elenco molto rapido di pro e contro:

Famiglia HAP

Pro

- Può riprodurre risoluzioni e frame rate estremamente elevati;
- Costo sulla CPU molto basso;
- Il formato HAP Q è privo di perdite a livello visivo;
- Costo sulla GPU molto basso;

Contro

- File molto grandi;
- Difficile codificare questi file su Windows;
- Necessità di utilizzare SSD o un RAID di SSD per la riproduzione;
- Il collo di bottiglia principale è la velocità di lettura dei dischi.

H.264

Pro

- File estremamente leggeri e di piccole dimensioni;
- Migliore rapporto tra qualità e dimensione dei file;
- Basso utilizzo dei dischi;

Contro

- Richiede un alto numero di core della CPU per riprodurre risoluzioni e frame rate molto elevati;
- Può presentare quantizzazione dei colori se non si presta attenzione durante la codifica;
- Il bitrate dipende fortemente dal contenuto;
- Limite di 4096 pixel nella risoluzione sia verticale che orizzontale;
- Difficoltà nel creare canali alfa.

Animation

Pro

- Qualità al 100% senza perdite;
- Priorità alla qualità;
- Supporto nativo ai canali alfa;

Contro

- File di grosse dimensioni;
- Esigente sia sui dischi che sulla CPU;
- Il bitrate fluttua con la quantità di dettagli nel contenuto del video.

Cineform

Pro

- Bitrate costante;
- Alta qualità dell'immagine;
- Supporto nativo ai canali alfa;

Contro

- Dimensione dei file;
- Occorre acquistare software apposito per la codifica Cineform.

4 CHOP

4.1 Introduzione

La famiglia degli Operatori Canale, o CHOP, gestisce tutte le operazioni sui canali, inclusi i dati da sensori di movimento, gli input audio, le animazioni con key-frame, gli ingressi hardware (da Microsoft Kinect, Leap Motion, Oculus Rift, penne dei tablet, tastiere, mouse, etc), DMX, MIDI e OSC. Sono gli Operatori che gestiscono l'ingresso, l'elaborazione e l'uscita dei dati utilizzati per comunicare con molti tipi di attrezzature audio-video, come ad esempio:

- Mixer;
- Controller MIDI;
- Sintetizzatori;
- Impianti luci DMX;
- Telecamere Microsoft Kinect;
- Computer che eseguono TouchDesigner;
- Casse;
- Altre applicazioni audio-video, come Max/MSP, Ableton Live, Resolume Arena.

4.2 Metodi di Comunicazione

Il protocollo MIDI funziona con un'enorme quantità di hardware e software: tutte le Digital Audio Workstation, o DAW, come Ableton Live, Avid Pro Tools, Propellerhead Reason e altre, supportano ingressi e uscite MIDI. È un protocollo relativamente veloce, stabile e rodato nel tempo. I controller per le performance oggi implementano spesso il protocollo MIDI attraverso le porte USB; questi controller includono vari tipi di input hardware, come pulsanti, fader, tastiere, strisce touch, jog wheel, pad e potenziometri.

Gli ambienti di programmazione come Cycling 74 Max/MSP, PureData, Native Instuments Reaktor e altri, supportano anche i messaggi OSC. Questi sfruttano i vantaggi della moderna tecnologia di rete, hanno una risoluzione maggiore rispetto al protocollo MIDI, la possibilità di assegnare una nomenclatura ai canali e molti altri miglioramenti strutturali. I messaggi OSC possono essere trasmessi attraverso connessioni UDP o TCP, rendendo incredibilmente semplice creare reti e trasmettere segnali in tempo reale anche su lunghe distanze. Al momento il protocollo OSC è il metodo di comunicazione tra i software e i computer maggiormente utilizzato.

Il protocollo DMX è utilizzato dagli impianti luci e dai loro controller. Le attrezzature DMX contano vari canali per i dimmer, diverse possibili impostazioni, tracciamenti predefiniti, canali RGB, automazioni per i motori e altro. Molti controller e banchi luci usano il protocollo DMX per comunicare con apparecchi e sistemi video computerizzati ma, dati i molti differenti tipi di controller e banchi disponibili, i loro manuali sono importantissimi quando si vogliono creare progetti che li coinvolgano. In generale, tutti i canali di un apparecchio devono essere considerati, anche se non sono effettivamente utilizzati. Ci sono molti metodi per ottimizzare il flusso di lavoro con i dati DMX, la maggior parte dei quali riguarda una buona gestione ed organizzazione dei canali, e verranno analizzati in esempi più avanzati.

I CHOP "Sync In" e "Sync Out" sono utilizzati per sincronizzare i frame di istanze differenti, sia interne che esterne, di TouchDesigner, sfruttando il protocollo OSC per la loro comunicazione. Questi due Operatori lavorano comunicando lo stato di ogni frame su ognuna delle macchine sincronizzate, e una volta che tutte le macchine confermano di avere renderizzato il fotogramma corrente, si spostano simultaneamente al frame successivo. Questa sequenza di eventi è ripetuta per ogni frame, e mantiene le macchine sincronizzate sempre sullo stesso fotogramma.

4.3 Ingressi e uscite Audio

L'audio può essere elaborato a partire da una varietà di sorgenti diverse, e può essere processato in molti modi differenti. TouchDesigner è in grado di elaborare il suono a partire da file audio, file video, interfacce audio esterne, sorgenti audio via internet, e può persino sintetizzarlo dal nulla.

La maggior parte dei progetti che coinvolgono sound design e tracce audio includeranno anche file dedicati: TouchDesigner è capace di leggere ed eseguire molti formati audio standard, come MP3, AIFF e WAV, attraverso l'uso del CHOP "Audio File In" e del CHOP "Audio Play"; questi file possono essere messi in loop, avviati, cambiati di pitch e tagliati, consentendo utilizzi flessibili di campioni e file audio.

Il CHOP "Audio Movie" può invece essere usato per riprodurre l'audio da un file video: anziché leggere l'audio facendo riferimento ad un file, questo CHOP fa riferimento ad un TOP "Movie In". È molto utile perché permette di mantenere l'audio sincronizzato con il video riprodotto dal TOP "Movie In", ed ha un parametro utile per spostare l'audio così da sincronizzarlo meglio con il video.

Esistono molte interfacce audio esterne che possono essere utilizzate con TouchDesigner, per una lista più comprensiva dei dispositivi compatibili è meglio fare riferimento alla Wiki di Derivative ed al Forum.

Questi dispositivi forniscono ingressi ed uscite audio analogiche e digitali. Gli ingressi possono provenire da musicisti e strumentisti, console di mix audio, videocamere professionali, computer e molto altro, mentre le uscite possono essere indirizzate verso molte delle destinazioni appena citate, così come verso sistemi di altoparlanti. I CHOP usati per comunicare con le interfacce audio esterne sono il CHOP "Audio Device In" ed il CHOP "Audio Device Out". Ognuno gestisce rispettivamente ingressi ed uscite, da e verso un progetto. Esiste anche un CHOP chiamato "Audio SDI", utilizzato in congiunzione con le schede nVidia Quadro SDI per ricevere audio da sorgenti SDI esterne.

Ci sono due diversi driver audio accessibili da TouchDesigner: il primo si chiama DirectSound, è disponibile fin dalle versioni precedenti di TouchDesigner ed è stato sviluppato come parte delle librerie DirectX. Si tratta di un driver maturo, sviluppato per molti anni, e rende disponibili latenze relativamente basse anche sotto carichi pesanti.

ASIO è invece una nuova aggiunta a TouchDesigner 088. È stato sviluppato da Steinberg per migliorare uno dei principali difetti delle DirectX, cioè il dover passare tutto l'audio attraverso il sistema operativo di Windows. Bypassando il sistema operativo, il driver ASIO è capace di comunicare direttamente con l'hardware audio esterno, fornendo quindi minori latenze di quanto possibile con DirectSound.

Una volta che gli ingressi e le uscite sono stati impostati, possono essere gestiti in pratica come qualsiasi altro flusso di dati.

4.4 Frequenze di Campionamento

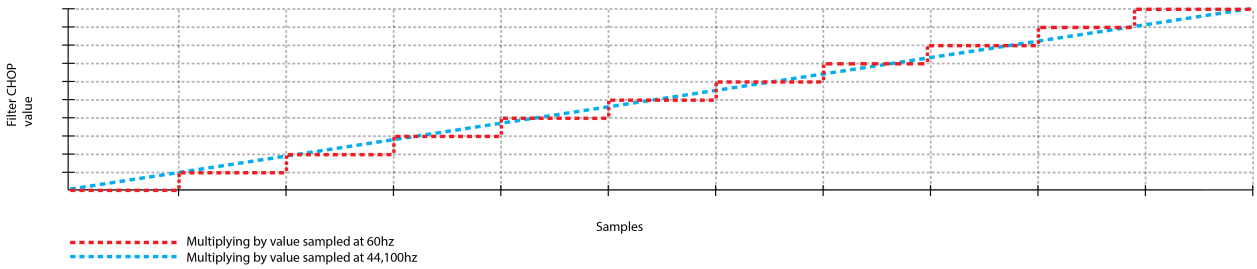
Molte applicazioni non mostrano mai le funzioni e le operazioni che svolgono dietro le quinte, perciò molte persone non sono abituate a considerare l'audio in maniera matematica. L'audio è, in sostanza, un flusso di dati numerici che vengono processati in modo incredibilmente veloce. Sapere ciò fornisce le fondamenta per lavorare con l'audio in TouchDesigner.

Aprite l'esempio "Sample_rates_1.toe". Questo progetto crea una funzionalità molto semplice, comune in molte applicazioni audio: mutare una traccia. Ciò è ottenuto utilizzando un CHOP "Math" che moltiplica il flusso audio per il valore emesso da un COMP "Button", che è 0 o 1. Come in ogni altra espressione matematica, un valore, in questo caso ogni campione audio, moltiplicato per 0 darà sempre 0; allo stesso modo, un valore, o un campione audio, moltiplicato per 1 sarà restituito immutato. Questi due valori producono gli stati on e off di questo esempio.

Spingiamoci un passo più avanti, permettendo al pulsante di far sfumare l'audio. Aprite il progetto "Sample_rates_2.toe".

Questo esempio prende quello precedente ed aggiunge due Operatori: il primo è il CHOP "Filter", che addolcisce il valore in ingresso, creando una rampa abbastanza lenta tra i due stati del pulsante; il secondo è un CHOP "Resample".

La frequenza di campionamento di diversi Operatori è sottovalutata da molti principianti, ma è essenziale per ottenere un'uscita audio priva di artefatti. Il CHOP "Oscillator" viene campionato 44100 volte al secondo, mentre il CHOP "Filter" è campionato 60 volte al secondo. Questa discrepanza significa che, quando vengono moltiplicati, non si avrà un rapporto 1:1 tra i campioni dell'audio ed i campioni della rampa; più precisamente, ci sarà un rapporto di 735:1. Ciò significa che, quando i due valori sono moltiplicati, l'audio salirà o scenderà di volume ogni 735 campioni. Esaminate il grafico seguente, dove la linea tratteggiata blu descrive un rapporto 1:1, mentre la linea tratteggiata rossa rappresenta un rapporto 735:1.



Osservando la figura si nota una discretizzazione molto distinta che avviene quando vengono moltiplicati i due canali con frequenze di campionamento diverse. Molti CHOP usano gli FPS del progetto come loro frequenza di campionamento di default, ingigantendo la discretizzazione quando il progetto è impostato a 30 FPS: utilizzando lo stesso esempio precedente, il rapporto tra i campioni audio ed i campioni della rampa passerebbe da 735:1 a 1470:1. Ciò significa che in un progetto a 30 FPS, avverrebbe un incremento del volume solamente ogni 1470 campioni!

L’esempio citato evidenzia la necessità di essere sempre coscienti delle frequenze di campionamento dei CHOP, e di saper ricorrere al CHOP “Resample” quando necessario. Il più delle volte riguarderà l’audio, ma ci sono situazioni in cui un flusso dati di controllo potrebbe avere bisogno di essere letto o trasmesso con frequenze di campionamento differenti da quelle originarie.

4.5 Tagliare il Tempo

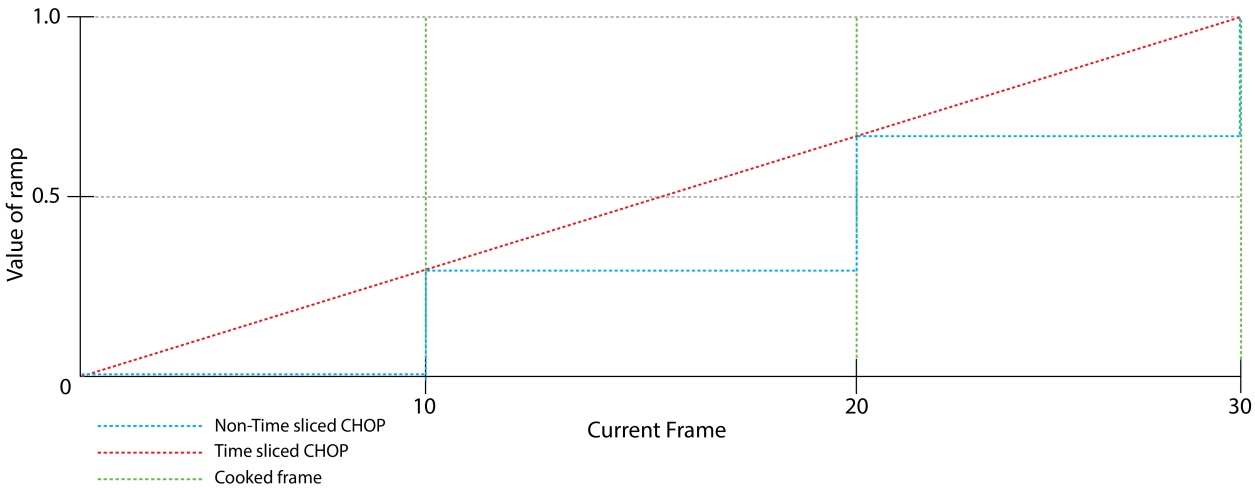
Tagliare il tempo è una caratteristica unica di TouchDesigner, e all'inizio può risultare un po' ostica da comprendere.

Un taglio di tempo è il periodo tra l'ultimo frame renderizzato ed il frame corrente. Pensate ai tagli di tempo come a quantità di tempo dinamiche: se un progetto va a 60 FPS costanti, allora il taglio di tempo sarà lungo 1 fotogramma; se un progetto ha delle difficoltà a rimanere al passo con il tempo reale, e perde 10 frame, il taglio di tempo corrispondente sarà lungo 10 frame.

I tagli di tempo esistono per regolare i dati dei CHOP in situazioni in cui vengono persi dei frame. Tagliare il tempo significa, semplificando, che i CHOP iniziano a prendere in considerazione la misura dei tagli di tempo durante il loro funzionamento. Immaginatelo come una specie di funzionamento adattivo: quando i tagli di tempo crescono in lunghezza i CHOP compenseranno per il numero di frame persi e calcoleranno il numero di fotogrammi necessari per produrre segnali più regolari. Ciò si contrappone ai CHOP che non tagliano il tempo, che si limitano a calcolare il loro valore all'ultimo frame elaborato, per poi saltare al valore del successivo fotogramma renderizzato, indipendentemente da quanti frame abbiano perso nel frattempo. Solo i CHOP possono avvantaggiarsi dei tagli di tempo.

Sfruttando ancora l'esempio precedente, quando la timeline sta andando costantemente a 30 FPS ogni taglio di tempo è lungo 1 frame. Se ci fossero due rampe che vanno da 0 a 1 nel corso di un secondo (30 frame), entrambi i segnali in uscita sarebbero rampe regolari. Se invece, per qualche bizzarro motivo, si elaborasse solo un frame ogni 10, si avrebbero risultati ben diversi: nel CHOP senza tagli di tempo il valore cambierebbe drasticamente ad ogni frame calcolato, mentre i dati compresi tra questi frame sarebbero persi; il CHOP con i tagli di tempo sarebbe invece conscio di essere eseguito solamente ogni 10 frame, e calcolerebbe i frame intermedi per interpolare tra il valore dell'ultimo frame elaborato ed il frame corrente. Ciò mantiene il flusso di dati regolare indipendentemente da quello che succede.

La figura seguente illustra l'esempio precedente: la linea blu tratteggiata rappresenta un CHOP senza tagli di tempo, mentre la linea rossa tratteggiata indica un CHOP con i tagli di tempo, e le linee verticali verdi tratteggiate sono i frame elaborati.



Time Slicing

4.6 Operatori Canale Più Diffusi

Questa sezione vuole introdurre alcuni degli Operatori Canale maggiormente utilizzati. Nella cartella .zip è incluso un file di esempio.

CHOP Generatori

CHOP Constant

Il CHOP "Constant" può contenere fino a 40 valori costanti. Il nome di ogni canale è definito nella casella di testo a sinistra, mentre il suo valore può essere selezionato utilizzando lo slider a destra.

Quando un CHOP "Constant" viene creato ha un solo canale attivo, mentre tutti i canali inattivi appaiono colorati di grigio; per attivare un canale è sufficiente dargli un nome: così il valore numerico si colorerà, e verrà mostrato nel viewer dell'Operatore.

Per creare più canali con lo stesso valore potete nominarli utilizzando dei pattern. Se inserite "Chan[1-4]" nella campo del nome verranno creati 4 canali: "chan1", "chan2", "chan3" e "chan4", tutti con lo stesso valore.

CHOP Noise

Questo CHOP crea un insieme di punti pseudo-casuali, distribuiti in base alle impostazioni selezionate nei parametri dell'Operatore. Esistono sei diversi algoritmi tra cui scegliere, ognuno con caratteristiche diverse, e ognuno adatto per determinate situazioni. La base di ogni algoritmo è il valore "seed": potete creare due CHOP "Noise" con gli stessi valori e lo stesso aspetto, ma cambiando il parametro "seed" genererete due insieme di punti differenti. *(es.1 in noise.toe)*

Per creare movimento potete cambiare i valori nella sezione "Transform". Se per esempio inserite "absTime.frame" nella prima casella del campo chiamato "Translate", potrete vedere i valori scorrere lungo l'asse x. *(es. 2 in noise.toe*)

È possibile creare più canali andando nella pagina "Channel" dei parametri, e inserendo nel campo "Channel Names" i nomi dei vari canali, separati da uno spazio. *(es.3 in noise.toe)*

Il numero di campioni che volete generare può essere impostato sempre dalla pagina "Channel", cambiando i valori "Start", "End" e "Sample Rate". Se però avete bisogno di un solo valore alla volta, potete andare nella sezione "Common" e mettere su "On" il parametro "Time Slice". Questo crea un valore casuale per ogni canale ad ogni frame, richiedendo meno utilizzo della CPU. *(es.4 in noise.toe)*

CHOP Pattern

Il CHOP "Pattern" genera una specifica funzione con un determinato numero di campioni; è possibile impostare la dimensione di questo insieme dal parametro "Length", nella pagina "Pattern", mentre il tipo di funzione può essere scelto alla voce "Type".

Il parametro “Cycles” indica invece il numero di ripetizioni della funzione all’interno del numero di campioni.

Sono presenti varie impostazioni per controllare il vostro pattern, a seconda del “Type” scelto.

Anche i parametri “From Range” e “To Range” possono rivelarsi molto utili: per esempio quando avete una funzione sinusoidale che genera valori compresi tra “-1” e “1”, ma a voi occorre un valore tra “0” e “1” (maggiori dettagli su questo nella sezione dedicata al CHOP “Math”). *(es.3 in pattern.toe)*

Il CHOP “pattern” è inoltre un importante strumento per creare tabelle di riferimento, le cosiddette lookup table. *(es.4 in pattern.toe)*

CHOP LFO

Il CHOP “LFO” genera un valore che oscilla secondo i criteri scelti : si sposta avanti e indietro tra due valori determinati da “Amplitude” e “Offset”, in un tempo selezionabile dalla voce “Frequency”.

“Frequency” generalmente determina quanti sono i cicli per secondo, eccetto quando è connesso un valore nel primo input del CHOP “LFO”, chiamato “Octave Control”. Se infatti “Octave Control” assume il valore “1”, la velocità viene raddoppiata, mentre se è impostato a “2”, la velocità è raddoppiata due volte (x4), e così via. *(es.2 in noise.toe)*

La forma delle oscillazioni è controllata da “Type”.

Potete anche far oscillare un pattern differente utilizzando il terzo ingresso dell’Operatore. *(es.3 in lfo.toe)*

CHOP Timer

Questo CHOP è molto utile per qualsiasi cosa coinvolga dei periodi di tempo fissati. Potete attivare il timer e ricevere diversi tipi di dati durante la sua durata, così come impostare un comportamento diverso quando termina.

Il timer può scorrere e fermarsi, scorrere e resettarsi al valore iniziale, oppure ripetersi un numero determinato o infinito di volte.

Utilizzate il parametro “Time” per impostare la durata del timer, e attivatelo per avviarlo o resettarlo. Potete usare “Delay” per inserire una certa attesa tra l’attivazione del pulsante “Start” e l’effettivo avvio del timer.

“Cue Point” crea un punto di riferimento all’interno del timer a cui potete saltare attivando il tasto “Cue”. Se per esempio voleste essere in grado di saltare direttamente ad un punto a metà della durata del timer, basterebbe impostare “Cue Point” a “0.5”.

Nella sezione “Outputs” potete selezionare quali informazioni volete ricevere. Alcuni dati comunemente utilizzati sono:

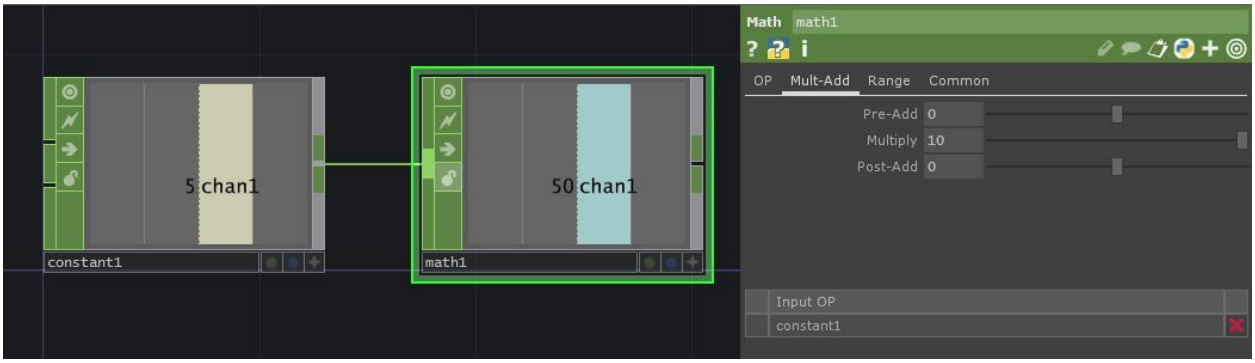
- `Timer Fraction`, che mostra la percentuale trascorsa del periodo di tempo fissato, in una scala da 0 a 1. Il valore 0.5 significherebbe che il timer è a metà, mentre 1 che ha terminato;
- `Done`, che è “0” quando il timer è inizializzato o sta scorrendo, e diventa “1” quando ha terminato. Tornerà “0” quando vengo nuovamente attivati i pulsanti “Init” o “Start”.

CHOP Filtri

CHOP Math

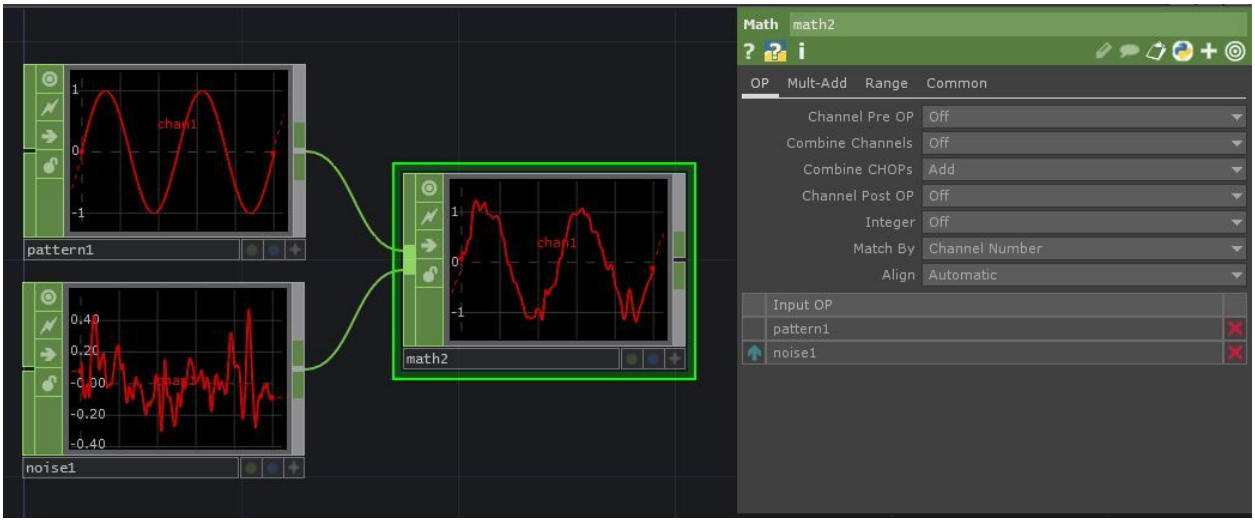
Questo è probabilmente il CHOP utilizzato più comunemente. Prende i dati dal suo ingresso e li manipola in diversi modi:

l'utilizzo più intuitivo sarebbe prendere un valore e compiere semplici operazioni matematiche, come aggiungere 10 o moltiplicare il valore dato per 10, possibile dalla sezione "Mult-Add".



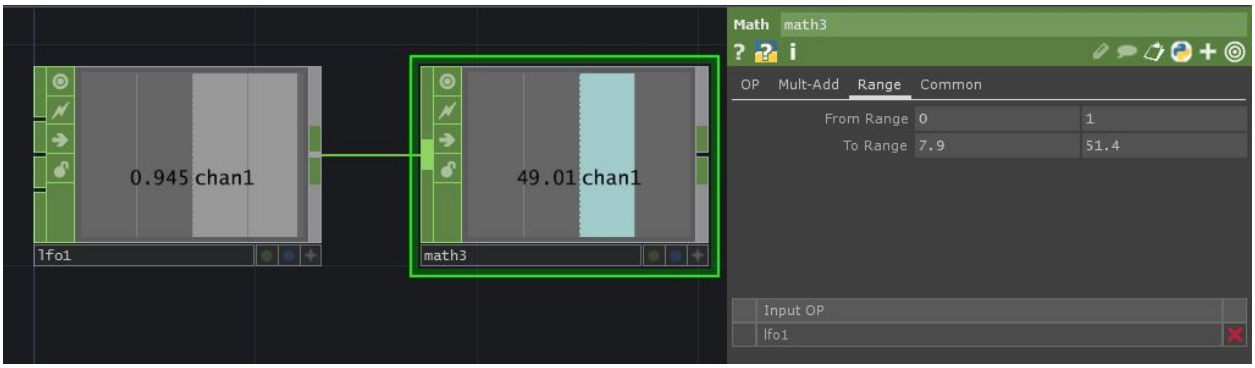
Order of Operations

Comunemente occorre prendere un valore, o un insieme di valori, e regolarli in accordo con un altro valore o un altro insieme di valori. Per esempio, se il nostro scopo finale è quello di ottenere un valore che si sposta su e giù nel corso del tempo, ma vogliamo anche aggiungergli un movimento casuale, possiamo usare un CHOP "Pattern" per creare un'onda sinusoidale, un CHOP "Noise" per generare un insieme di numeri casuali, e sfruttare il parametro "Combine CHOPs" impostato su "Add" all'interno della sezione "OP" di un CHOP "Math". Il risultato dovrebbe essere simile a questo:



Combine Chops

Un'altra funzione molto utile del CHOP "Math" è nella pagina "Range". Qui è possibile prendere un insieme di valori e rimapparli in un nuovo intervallo. Per esempio, se avete un CHOP "LFO" che crea delle rampe da 0 a 1, ma volete rimappare gli stessi campioni tra i valori 7.9 e 51.4, è molto più facile utilizzare i parametri "From Range" e "To Range" che non impostare varie operazioni successive.



Range

CHOP Select

Questo CHOP può essere utilizzato per dividere i dati contenuti all'interno di un singolo CHOP, o per ricevere dati da una parte distante del progetto. Contemporaneamente potete anche rinominare i canali che state selezionando.

Se avete un CHOP con diversi canali, e avete bisogno di svolgere delle operazioni solo su alcuni dei suoi canali, un CHOP "Select" vi permetterà di scegliere i canali su cui concentrarvi. Utilizzate il campo "Channel Names" nella pagina "Select" dei parametri per elencare i canali che vi servono, separandoli con uno spazio.

Se volete rinominare questi canali in qualcos'altro, potete usare il campo "Rename From" per scrivere i nomi originali, ed il campo "Rename To" per inserire i nuovi nomi.

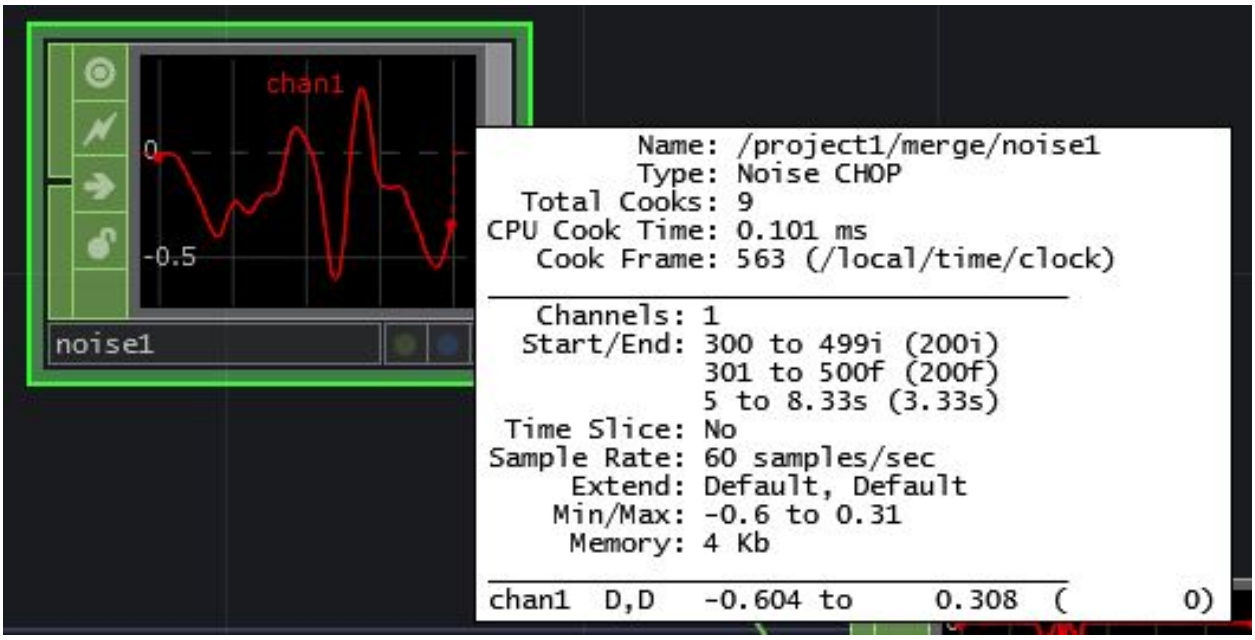
Questo CHOP vi permette anche di fare riferimento ad altri CHOP da differenti aree del progetto. Ipotizziamo che abbiate dei dati in un CHOP "Constant" all'interno di altri due COMP. Il percorso del CHOP "Constant" è "/project1/base1/base1/constant1", ma il vostro progetto deve far riferimento a dei dati nel vostro COMP "/project1". Potete creare dei CHOP "Out" e collegare manualmente le informazioni, oppure utilizzare un CHOP "Select" per fare riferimento senza collegare fili al percorso del CHOP "Constant", in modo più veloce e mantenendo il progetto meglio organizzato. Nel campo "CHOP" della sezione "Select" dei parametri, inserite "/project1/base1/base1/constant1", e sarete in grado di vedere i dati.

Come prima, se volete selezionare solo un canale all'interno del CHOP "Constant", potete utilizzare il campo "Channel Names" per selezionare quello desiderato, ed eventualmente rinominarlo.

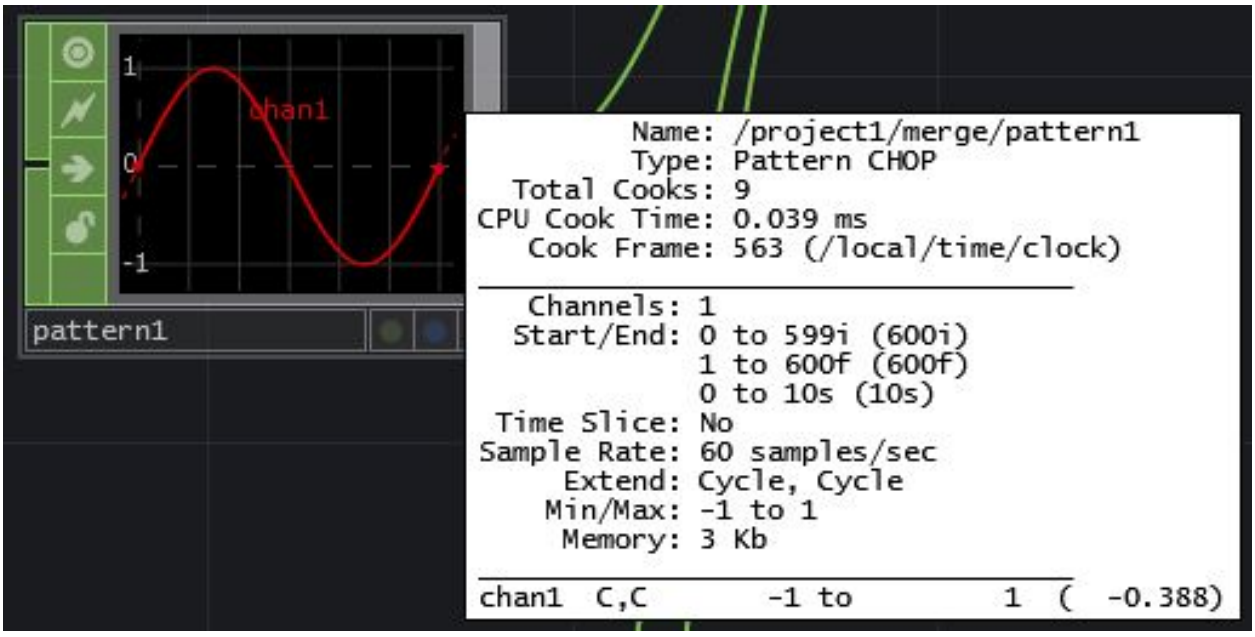
CHOP Merge

Il CHOP "Merge" è l'opposto del CHOP "Select": prende canali da CHOP diversi e li unisce in un unico CHOP.

L'idea è piuttosto semplice, ma se i risultati sono diversi da quelli che vi aspettavate, conviene controllare se nei CHOP in ingresso i campioni "Start/End" corrispondono, facendo click con il tasto centrale del mouse.



Range



Range

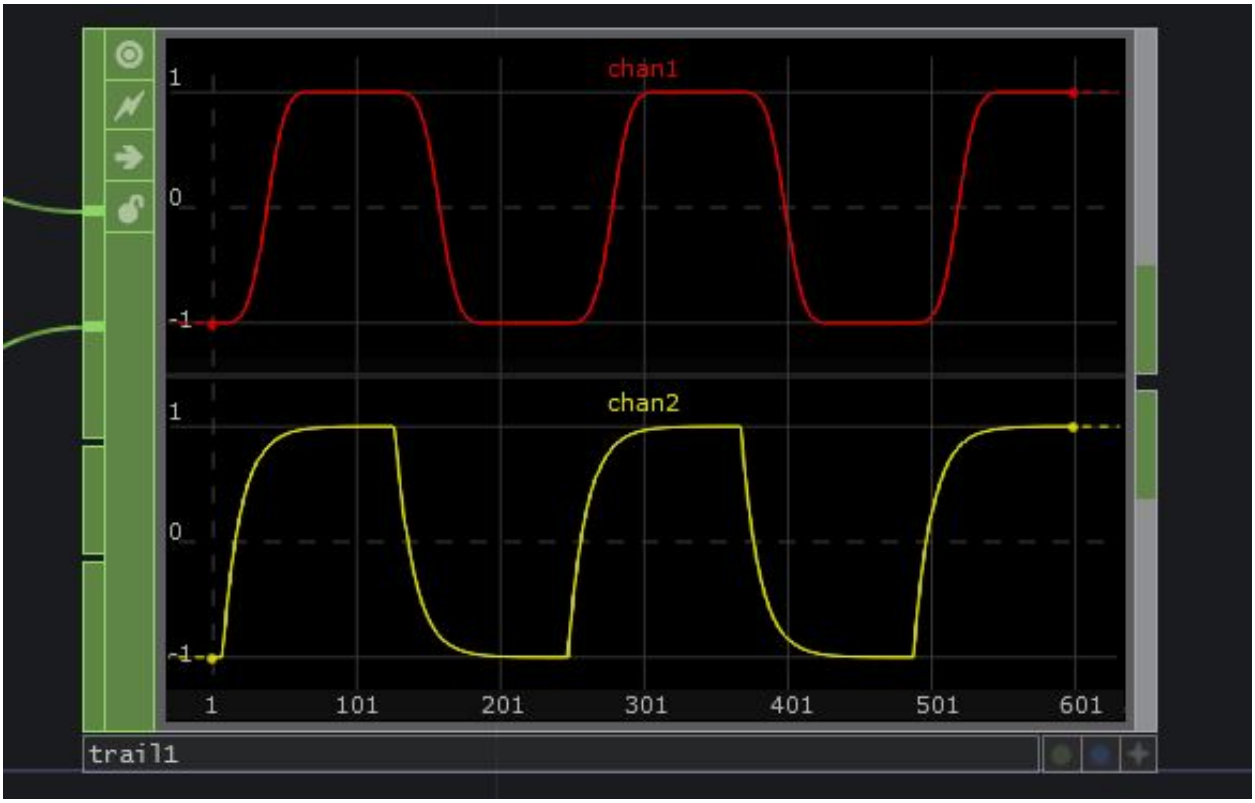
Nei CHOP appena mostrati, sia i campioni iniziali che quelli finali sono differenti. Si può porre rimedio a ciò impostando le "Extend Conditions" nella sezione "Channel" dei parametri dei CHOP in ingresso, oppure con le opzioni di "Align" nei parametri del CHOP "Merge".

Esiste anche una spiegazione dei diversi metodi di estensione nella pagina della Wiki relativa al CHOP “Extend”, che si può trovare qui[3]

Potete anche aprire il progetto di esempio e sperimentare con le diverse “Extende Conditions” e le varie opzioni del campo “Align”.

CHOP Trail

Il CHOP “Trail” crea una visualizzazione di come cambino nel tempo i valori che riceve in ingresso. Può rivelarsi molto utile quando occorre vedere piccole differenze nei movimenti dei canali, o osservare come cambino i valori di un canale rispetto a quelli di un altro.



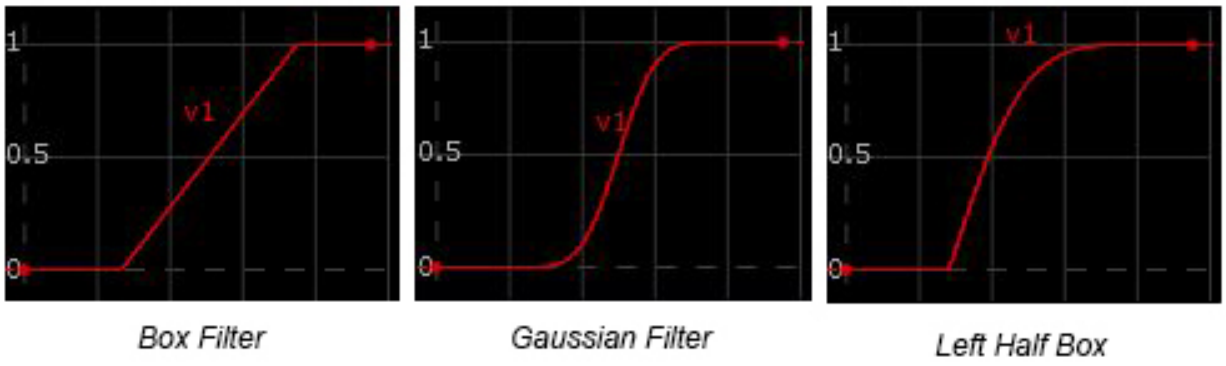
Trail

CHOP Filter / Lag

I CHOP “Filter” e “Lag” creano transizioni graduali tra due valori in un tempo definito. I due CHOP hanno scopo simile ma opzioni diverse.

Il CHOP “Filter” applica un effetto di smorzamento nel tempo, ed è possibile scegliere la forma dello smorzamento, con opzioni differenti per ogni forma.

[3] http://www.derivative.ca/wiki088/index.php?title=Extend_CHOP



Filter Types

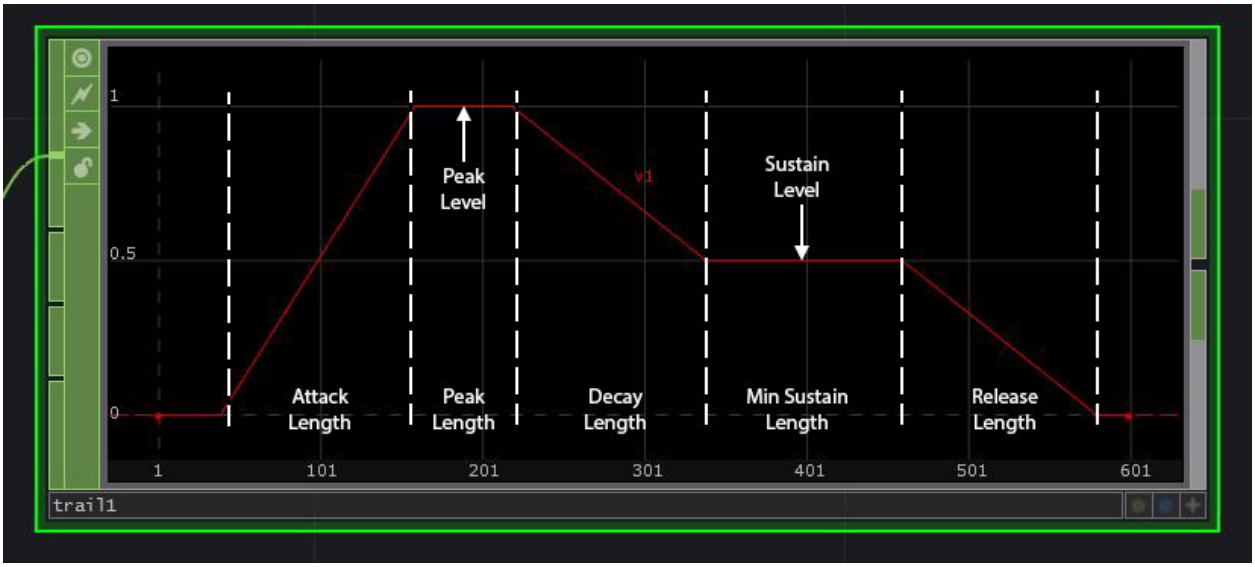
*(vedere **/filter_lag/ example 1** del progetto di esempio)*

Il CHOP "Lag", a seconda del metodo di smorzamento, permette di impostare due durate dell'effetto differenti, una per i valori crescenti e una per quelli decrescenti. *(vedere **/filter_lag/ example 2** del progetto di esempio)*

CHOP Trigger

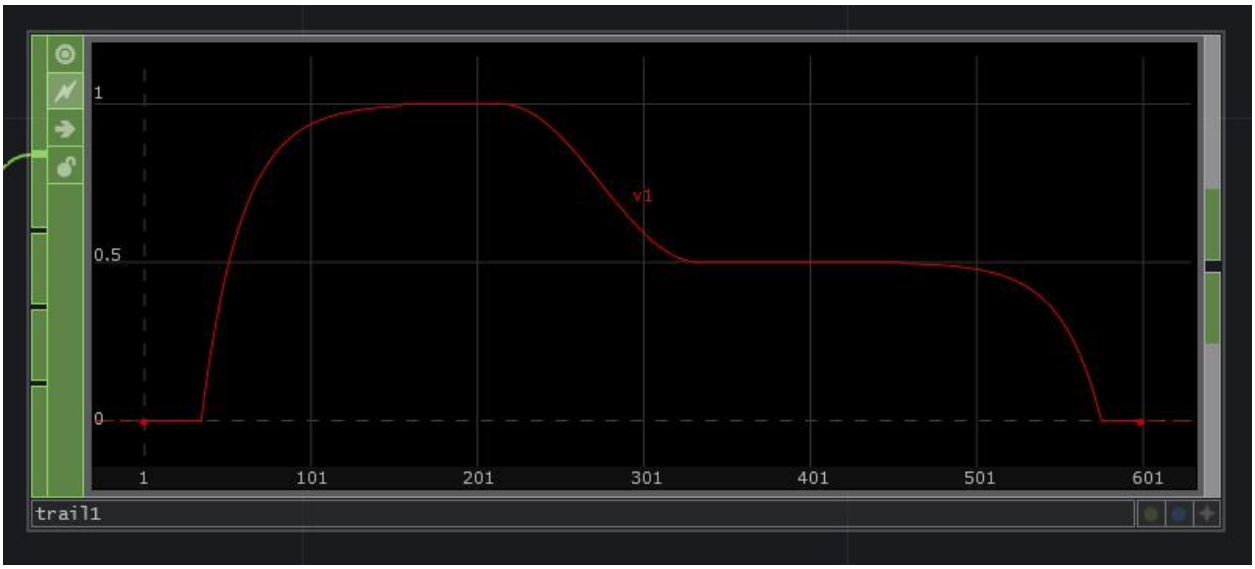
Questo CHOP prende un evento e crea un inviluppo ADSR (attack, Decay, Sustain, Release) con alcuni controlli addizionali.

L'inviluppo può essere attivato del "Trigger Pulse" nella pagina dei parametri "Trigger", oppure connettendo un CHOP al suo ingresso, per esempio collegando l'uscita di un COMP "Button". Esistono diverse sezioni dell'inviluppo:



Filter Types

Le diverse sezioni possono a loro volta avere delle attenuazioni. Di seguito un esempio con “Attack Shape” impostato su “Ease out”, “Decay Shape” su “Ease in Ease out” e “Release Shape” su “Ease in”:



Filter Types

5 DAT

5.1 Introduzione

Gli Operatori DAT compiono operazioni sui dati: possono modificare, analizzare, creare, inviare e ricevere flussi di dati in varie forme, spaziando fra stringhe di testo, tabelle, script Python, XML, JSON, MIDI, Seriali, OSC e molti altri.

Molti progetti fanno affidamento sull'utilizzo di DAT e script Python; essere perciò in grado di analizzare tabelle piene di informazioni e di metadati, controllare altri Operatori ed il loro stato, eseguire compiti complessi basati su segnali provenienti da altri sistemi ed altro, direttamente in TouchDesigner, rende possibile creare sistemi con logiche piuttosto complesse. Dato che tali sistemi sono una delle cose che rendono TouchDesigner unico, nella porzione di libro dedicata agli esempi saranno riportati diversi esempi.

Un modo interessante di pensare a TouchDesigner è come un ambiente modulare di programmazione in Python: immaginate di prendere lunghi e complessi programmi Python e di spezzarli in porzioni brevi e modulari, e di salvarli individualmente all'interno di DAT "Text". Un setup di questo tipo risulta di semplice lettura, è facile da mantenere, da espandere e, cosa persino più importante, facile da condividere per lavori collaborativi.

5.2 Metodi di Comunicazione

Esistono molti DAT capaci di sfruttare i diversi protocolli di comunicazione per fornire ingressi ed uscite. TouchDesigner è in grado di comunicare nativamente attraverso i protocolli MIDI, OSC, TCP, UDP, UDT e Websocket, con la possibilità di parlare a molti programmi differenti, applicazioni web e servizi, e di mostrare hardware di controllo, altri computer, eccetera.

I protocolli MIDI, OSC e DMX sono già stati spiegati nel capitolo sui CHOP; i DAT possono comunicare sui loro stessi protocolli, più alcuni altri.

TCP è il protocollo di comunicazione standard per internet. È un protocollo orientato alla connessione tra sistemi, presentando quindi una chiara relazione di client e server tra le parti messe in comunicazione, e deve essere stabilita una connessione (di solito nascosta) prima di trasmettere qualsiasi dato. Queste caratteristiche permettono alle connessioni TCP di essere molto affidabili, poiché le parti comunicanti possono fornire prova di tutti i dati inviati e ricevuti, evitando perciò qualsivoglia perdita di dati; sono inoltre dei flussi di dati ordinati, quindi i dati inviati in un certo ordine saranno ricevuti in quello stesso ordine.

UDP, al contrario, è un protocollo privo di connessione, e prima della trasmissione dei dati non viene stabilito nessun collegamento. Ciò crea un certo livello di inaffidabilità in aree come la prova di trasmissione, i pacchetti persi e l'ordine dei messaggi. Se nessuno di questi aspetti è cruciale per un progetto, nei sistemi per cui la perdita di pacchetti è meno importante di eventuali ritardi esistono alcuni vantaggi in termini di performance.

UDT è uno dei protocolli di comunicazione più recenti: essenzialmente è la combinazione delle parti migliori di UDP e TCP. Esso è costruito a partire dal protocollo UDP, ma è basato sulla connessione, e perciò molto affidabile. Ciò significa che ha le stesse prove di trasmissione e lo stesso ordinamento dei dati che avrebbe una connessione TCP, ma attraverso il più veloce protocollo UDP.

Un vantaggio dei protocolli privi di connessione è che essi supportano una caratteristica chiamata "Multicast messaging": indipendentemente da quanti computer siano collegati, ogni messaggio è trasmesso a tutta la rete una volta sola; quando occorre inviare lo stesso messaggio a molti computer, non è quindi necessario inviare lo stesso messaggio ad ogni sistema individualmente. È l'opposto del "Unicast messaging", in cui ogni computer connesso riceve un messaggio individuale, anche quando il messaggio inviato a tutte le macchine è lo stesso.

Websocket è invece il metodo di comunicazione da preferire quando si lavora con browser web e applicazioni online real-time. È stato disegnato per essere utilizzato come un'interfaccia tra i browser web ed i server, permettendo di semplificare alcune funzionalità fondamentali per le comunicazioni web bi-direzionali.

6 SOP

6.1 Introduzione

La famiglia degli Operatori di superficie, o SOP, è utile per tutte le operazioni 3D: questo include lavorare con semplici geometrie 3D, sistemi di particelle, modelli architettonici, personaggi 3D ed tanto altro. I SOP sono Operatori spesso trascurati da molti principianti a causa della loro ripida curva di apprendimento, ma una solida conoscenza del loro funzionamento apre molte opportunità interessanti per i progetti, e offre nuovi metodi estremamente efficienti per risolvere vari tipi di problemi.

Progetti che coinvolgono projection mapping, motion capture 3D in tempo reale, installazioni LED e video per facciate architettoniche sarebbero impossibili o estremamente difficili da realizzare senza la famiglia degli Operatori SOP.

TouchDesigner 088 al momento supporta i seguenti tipi di file 3D:

- .fbx
- .obj
- .3ds
- .dxf
- .dae

Per migliorare le performance è essenziale evitare che gli Operatori vengano calcolati quando non necessario, cosa che sarà discussa più avanti nel capitolo "Ottimizzazione". Ciò è ancora più importante quando riguarda i SOP. Cercate sempre di applicare le trasformazioni ai COMP "Geometry" invece che direttamente ad un SOP: le trasformazioni dei SOP avvengono infatti sulla CPU, le devono essere eseguite per ogni vertice presente nella geometria, occupando molte risorse; invece le trasformazioni a livello dei componenti sono applicate direttamente alla geometria 3D, o all'oggetto, nel suo intero e sono calcolate sulla GPU come una singola operazione. Una singola operazione compiuta sulla GPU è decisamente da preferirsi rispetto a quelle che potrebbero essere centinaia di migliaia di operazioni eseguite sulla CPU.

Il numero totale di punti, primitive, vertici e mesh varierà a seconda di quale modello si stia elaborando, ma il principio di base è che più poligoni/vertici sono presenti, più potenza di calcolo e memoria grafica saranno richiesti per completare le varie operazioni. Esistono strumenti all'interno di TouchDesigner per ridurre il numero di poligoni in modelli complessi, ma ottimizzare la geometria all'interno di programmi di modellazione dedicati può fornire maggiore flessibilità.

6.2 Rendering

Un processo con cui molti principianti faticano è il passaggio in modo veloce ed efficiente da un flusso di lavoro in 3D ad uno in 2D. Internet è pieno di tutorial sui workflow 3D che possono spiegare molti dei punti più sottili del rendering 3D, ma per questo capitolo l'obiettivo è andare da un punto A, un semplice oggetto 3D, ad un punto B, un Render TOP.

In una scena 3D sono presenti tre elementi distinti:

1. La geometria 3D (ed i materiali);
2. La telecamera;
3. L'illuminazione.

Aprite l'esempio "Rendering_1.toe". Già ad una prima occhiata si notano le tre parti fondamentali necessarie per renderizzare la scena, camera, illuminazione e geometria 3D, tutte collegate ad un TOP "Render". Analizziamo separatamente ogni aspetto del setup di rendering, e osserviamo poi come tutto si incastra insieme.

L'elemento più adatto per cominciare è la geometria 3D, il punto fondamentale del rendering. Il modello 3D può essere qualsiasi cosa: semplici poligoni, personaggi 3D animati, modelli architettonici o altro. Sia che si importino dei modelli o che li si crei proceduralmente, tutte le operazioni sono svolte con dei SOP che convergono in un COMP "Geometry". Il concetto chiave da assimilare è che in TouchDesigner i SOP non sono mai renderizzati direttamente, ma vengono renderizzati i COMP "Geometry", i componenti delle geometrie che contengono i SOP da renderizzare. Possiamo utilizzare due scenari per dimostrarlo.

Aprite l'esempio "Rendering_2.toe". In questo progetto è presente una singola geometria trasmessa a 4 diversi COMP "Geometry", ognuno dei quali è renderizzato con valori di trasformazione diversi. Come esempio questo progetto utilizza un SOP "Box", ma tutto quello che mostriamo è applicabile anche su modelli più complessi. Compiere operazioni su un modello molto complesso, presente in diverse iterazioni, potrebbe saturare rapidamente le risorse del sistema: tornando all'idea che i SOP non sono renderizzati direttamente diventa logico caricare il modello una sola volta, e crearne quindi alcune iterazioni sfruttando i COMP "Geometry".

Aprite ora l'esempio "Rendering_3.toe". Questo progetto è molto diverso da quello precedente, in quanto è presente un solo COMP "Geometry" contenente tre diversi modelli. Se teniamo a mente il fatto che vengono renderizzati solo i COMP "Geometry", è logico raggruppare questi modelli in un singolo componente: la differenza può sembrare arbitraria, ma quando i progetti diventano più complicati è fondamentale risparmiare risorse.

Fino adesso le varie geometrie non presentavano alcun materiale, ma proprio i materiali sono ciò che rende interessanti le scene 3D: la differenza tra un blocco di cemento e un cubo in 3D risiede proprio nell'utilizzo dei materiali. Le texture possono essere applicate su due livelli: a livello dei SOP e a livello del componente. Il primo metodo prevede l'uso del SOP "Material", mentre il secondo fa riferimento al materiale inserito nell'opzione "Render" di un COMP "Geometry". Nell'esempio

“Rendering_4.toe” entrambi i materiali appaiono uguali, ma ognuno utilizza un metodo diverso per applicare la texture.

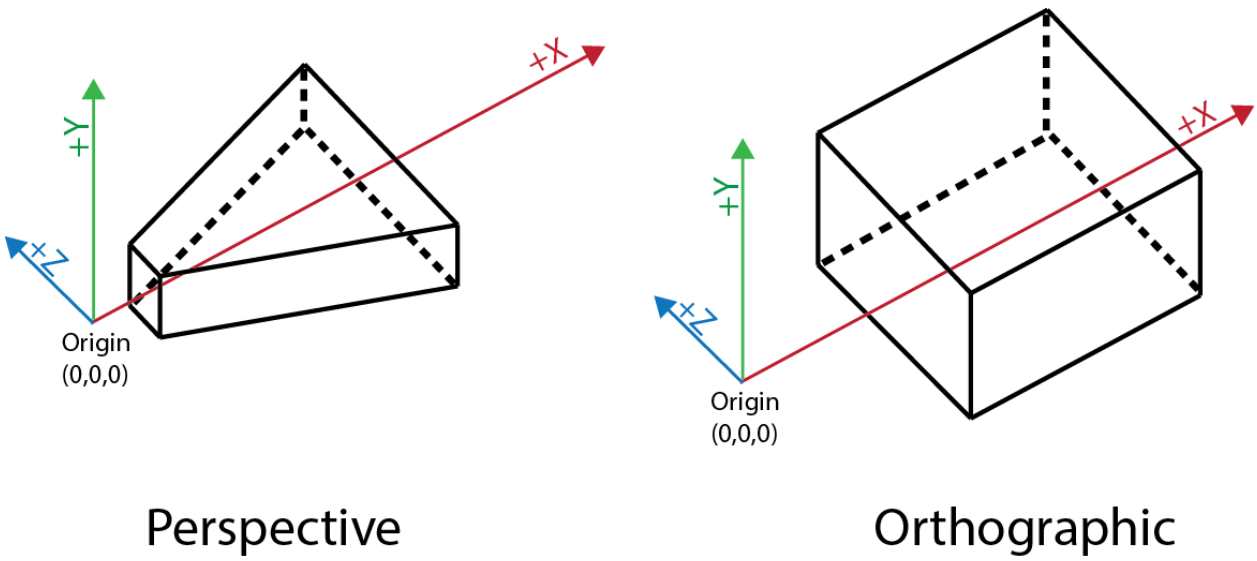
Ora che abbiamo aggiunto i materiali, discutiamo velocemente delle mappe UV. Aprite l’esempio “Rendering_5.toe”. Questo progetto è uguale a quello precedente, ma la stessa texture di prima appare ora completamente diversa: ciò è dovuto al fatto che le coordinate della mappa UV sono state modificate.

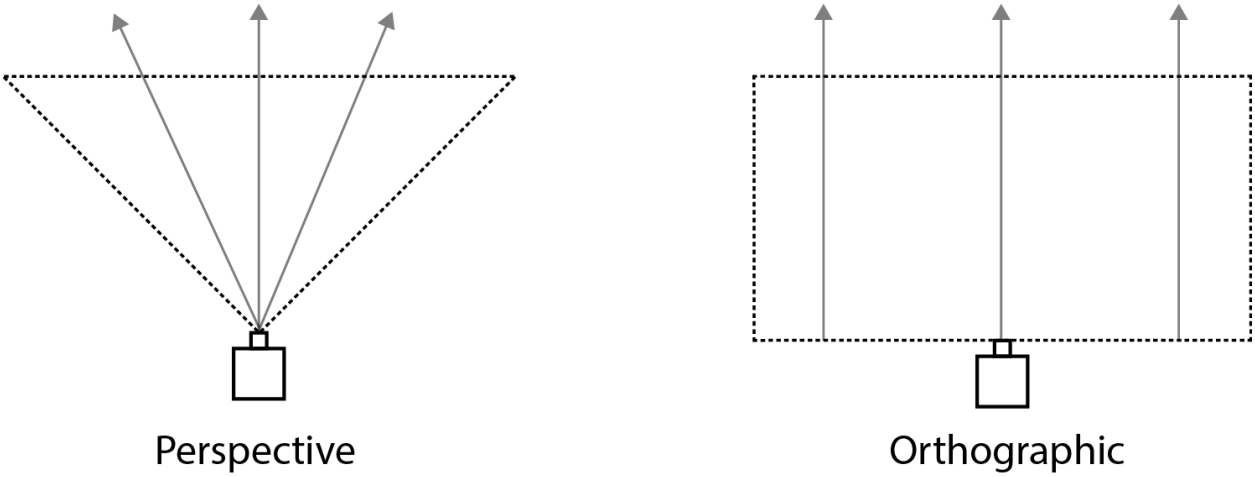
Il secondo elemento fondamentale di una scena 3D è la luce: come nella vita reale infatti le scene 3D necessitano di illuminazione. Aprite l’esempio “Rendering_6.toe”. Questo progetto mostra il render di un semplice cubo, ma niente è visibile nel TOP “Render”. Questo è dovuto al fatto che la luminosità della sorgente luminosa è stata intenzionalmente impostata a 0, per mostrare quanto sia importante l’illuminazione, e quanto spesso venga data per scontata.

I prossimi progetti di esempio presentano nuovamente la stessa scena, ma con luci differenti: in ognuno di questi progetti il COMP “Light” è stato trasformato per illuminare il SOP “Box” da angolazioni diverse. I progetti di esempio sono: “Rendering_7.toe”, “Rendering_8.toe” e “Rendering_9.toe”.

L’ultimo elemento utile a renderizzare una scena 3D è la telecamera, che fornisce sia il punto di vista che la prospettiva da utilizzare. Quello che la camera vede è esattamente ciò che verrà poi renderizzato. Aprite l’esempio “Rendering_10.toe”. Tutto il movimento nella scena deriva dall’animazione della posizione della telecamera. Molte volte una camera è inserita nella scena e dimenticata lì, il che può portare a scene 3D noiose, molto statiche e prive di vita. Non siate spaventati di pensare come un regista e sperimentate con la posizione, con le lunghezze focali e con i movimenti della telecamera.

Si utilizzano principalmente due tipi di telecamera: le camere prospettiche e quelle ortografiche.





Le camere prospettiche necessitano di ben poche spiegazioni, dato che lavorano in modo simile all'occhio umano: hanno un centro prospettico e un cono visivo che vengono utilizzati per determinare cosa vede la telecamera e come ciò debba essere renderizzato. Immaginate tutta la luce della scena 3D che si incanala verso il centro prospettico. La correzione data dalla prospettiva, o scorcio, è applicata a tutti gli oggetti nella scena, implicando che gli oggetti più lontani dalla telecamera appaiano più piccoli. Questo tipo di telecamera è quella utilizzata la maggior parte delle volte, dato che funziona in modo simile all'occhio umano. Un esempio di questo tipo di camera può essere trovato nel file "Camera_1.toe".

Questo progetto mette in evidenza la correzione prospettica. Nella scena sono presenti due cubi, entrambi della stessa dimensione, e il cubo posizionato più lontano dalla telecamera appare più piccolo, proprio come avverrebbe nella vita reale. Questo è l'esempio più semplice di come funziona una camera prospettica.

Le camere ortografiche, al contrario, sono molto diverse. Chi è famigliare con software CAD (computer-aided design) potrebbe avere incontrato questo tipo di telecamere. Il principio chiave che ne regola il funzionamento è il fatto che non abbiano un singolo punto prospettico, e dunque tutta la luce dell'immaginaria scena 3D non filtrerebbe in un singolo punto, come in una camera prospettica. I modelli non sono distorti dalla loro posizione nello spazio, e perciò non importa quanto lontano dalla camera sia un oggetto, esso non apparirà né più grande né più piccolo degli altri oggetti nella scena.

Aprite l'esempio "Camera_2.toe". Questo progetto è esattamente lo stesso di "Camera_1.toe", ma con una grande differenza in ciò che viene renderizzato: in questo esempio i due cubi sembrano essere uno di fianco all'altro, indipendentemente dal loro posizionamento nello spazio. Può essere un concetto difficile da comprendere la prima volta che viene presentato. Ci possono essere due modi di figurarselo:

Primo Immaginate una scena 3D che viene schiacciata su un singolo piano prima di essere renderizzata. Immaginate se il videogioco originale di Super Mario fosse un effettivo scenario 3D, renderizzato utilizzando una telecamera ortografica: non importa dove sono nello schermo i nemici e le piattaforme, se sul bordo o in centro, essi appaiono sempre della stessa dimensione e forma e

non subiscono alcuna correzione prospettica.
 Secondo Pensate ai progetti di un edificio in un programma CAD: è una rappresentazione su un piano 2D di un oggetto 3D. Non importa come viene percepito il progetto sullo schermo, 1 unità di misura sul bordo dello schermo è esattamente la stessa lunghezza di una unità di misura nel suo centro. Ogni cambiamento nella posizione della telecamera non distorce ciò che viene renderizzato.

Questo comunque non significa che la profondità sia irrilevante quando si utilizza una camera ortografica: essa anzi diventa fondamentale quando si stratificano diversi pezzi di geometria in una scena.

La telecamera è ugualmente importante per i progetti di projection mapping. Questa tecnica sarà esplorata in esempi futuri, ma per adesso è importante capire un altro dei ruoli delle camere. Nel projection mapping lo scopo principale è trasformare oggetti del mondo reale in superfici 3D: per ottenere ciò si importa un modello 3D dell'oggetto in TouchDesigner e si applicano delle texture utilizzando una varietà di sorgenti; questo oggetto con texture necessita quindi di essere renderizzato. I COMP "Camera" entrano qui in gioco per simulare i proiettori reali: raccogliendo più informazioni possibili riguardo le caratteristiche di un proiettore e le sue lenti, il COMP "Camera" può infatti replicare accuratamente all'interno di TouchDesigner il punto di vista del proiettore, permettendo quindi al modello di venire renderizzato, emesso, calibrato e allineato con l'oggetto reale. È la base del projection mapping.

7 COMP

7.1 Introduzione

Esistono tre tipi diversi di Operatori Componente, o COMP, e ognuno ha impieghi differenti:

I componenti **Object** creano, illuminano e permettono di mostrare le scene 3D;

I componenti **Panel** creano componenti dell'interfaccia utente, come pulsanti, slider e pannelli delle finestre;

I componenti **Other** includono i componenti che registrano i keyframe di animazione, quelli che replicano altri Operatori e quelli che creano finestre di output.

Gli Operatori Componente si utilizzano generalmente insieme ad altri Operatori: gli "Object" sono usati in varie combinazioni per creare e renderizzare SOP e scene 3D; i componenti "Panel" per creare interfacce utente e vari contenitori per unire uscite diverse; i componenti "Other" per diversi compiti, come animazioni con keyframe, repliche dinamiche di altri Operatori, apertura delle finestre su vari display, eccetera.

Una caratteristica interessante per mettere le cose in prospettiva è che una grande parte di TouchDesigner è costituita da componenti interni a TouchDesigner stesso. Capire ciò aiuta ad afferrare la granularità di TouchDesigner, e come sia possibile approcciare lavori su vari progetti. Per esempio tutti i componenti "Panel" sono costruiti a partire da altri Operatori. Create un COMP "Button" e, all'interno del suo Network, potrete notare che il suo sfondo è creato con un TOP "Text", mentre i suoi valori di on/off sono generati da un CHOP "Panel". Tutta l'interfaccia utente di TouchDesigner è composta e salvata nel container "ui" nel livello alla base di tutti i progetti, e persino i menu e le finestre di dialogo, come la finestra "MIDI Mapper" e la finestra "Variables", sono creati sfruttando altri componenti di TouchDesigner.

7.2 COMP Window

Il COMP “Window” è utilizzato quasi in ogni progetto per mostrare il contenuto di un Operatore in una nuova finestra. Sia che si usi il COMP “Window” per creare un output a schermo intero, sia per creare un’applicazione in una finestra più tradizionale, esistono numerose opzioni che dovranno essere modificate di volta in volta. A causa della natura unica di ogni progetto infatti non ci sono “impostazioni migliori” su cui poter fare sempre affidamento. Investite del tempo per controllare il significato dei vari parametri sulla Wiki, e sperimentate per trovare ciò che funziona meglio in ogni nuova situazione.

Aprite l’esempio “Window.toe”. Questo progetto illustra una pratica molto utile, oltre ad alcune semplici funzionalità del COMP “Window”. È buona cosa utilizzare un COMP “Container” come sorgente dell’uscita del COMP “Window”, perché la texture di un TOP può essere erroneamente trascinata all’interno dello schermo, persino in Modalità Performance; se ciò avviene la texture rimarrà spostata finché non si ricaricherà il progetto, o finché non verrà spostata nuovamente nella posizione originaria. Lo stesso spostamento di texture non avviene in un COMP “Container” poiché di default non può essere trascinata, ed essa resterà quindi sempre coerente.

Le altre funzionalità mostrate in questo esempio sono piuttosto semplici. Il primo pulsante, collegato ad un CHOP “Null” a cui fa riferimento il parametro “Open in Separate Window” del COMP “Window”, permette facile accesso all’apertura della finestra. Il secondo pulsante invece controlla dinamicamente quanti monitor sono collegati, contando le file del DAT “Monitors”; usando quel valore per far variare un CHOP “Count” è possibile aprire la finestra con il primo pulsante, e utilizzare poi il secondo pulsante per spostare attraverso i monitor a cui è assegnata.

7.3 Componenti dell'Interfaccia Utente

Gli Operatori Componente sono incredibilmente importanti perché creano l'interfaccia utente in TouchDesigner: nello specifico i Componenti "Panels" sono ciò che permette questa funzione. Creeremo molte interfacce utente in esempi più avanzati, perciò in questa sezione esamineremo solamente pochi esempi basilari.

Tre dei COMP "Panel" più utilizzati sono:

- Il COMP "Slider";
- Il COMP "Button";
- Il COMP "Container".

I primi due hanno le stesse capacità degli slider e pulsanti presenti in altre applicazioni, ma possono essere modificati per rispondere a richieste differenti: i pulsanti possono essere programmati come pulsanti a due posizioni, come selettori o come interruttori momentanei; gli slider invece possono funzionare come potenziometri a singolo asse, o come pad XY completi.

I COMP "Container" invece non hanno alcuna altra funzione se non fungere da contenitori per altri elementi dell'interfaccia utente.

Aprite l'esempio "UI.toe". In questo progetto viene mostrata una semplice interfaccia utente: partendo dal fondo del Network sono presenti 2 COMP "Button" e 5 COMP "Slider", i componenti che creano effettivamente le funzionalità dell'interfaccia utente. L'Operatore genitore di questi elementi è utilizzato per raggruppare e distribuire i vari pulsanti e slider. Notate che se i visualizzatori di "container1" o "container2" sono attivati, gli elementi dell'interfaccia utente sono utilizzabili, anche se né "container1" né "container2" hanno alcuna uscita o Operatori nel loro Network. Il risultato è lo stesso quando "container1" e "container2" sono combinati all'interno di "container3". Questo perché i COMP "Container" hanno l'abilità di mostrare i propri figli nei loro visualizzatori, facilitando la creazione di complesse interfacce grazie ad una combinazione di elementi più piccoli.

8 MAT

8.1 Introduzione

Gli Operatori Materiali, o MAT, vengono utilizzati per creare i materiali e gli shader da applicare alle geometrie 3D.

Un'approfondita conoscenza della grafica digitale e dei software di rendering può aiutare molto quando si lavora con questa tipologia di Operatori; senza di essa molti utenti saranno limitati alle impostazioni base dei MAT "Phong" e "Point Sprite".

Applicare texture e mappe UV alle geometrie è un processo complesso, perciò i risultati migliori spesso si ottengono utilizzando pacchetti dedicati alla modellazione. Le geometrie così texturizzate possono quindi essere importate e processate con TouchDesigner.

Esistono molti strumenti per modificare le texture, come il SOP "Texture", ma i processi più complicati possono essere realizzati in modo più agile all'interno di software creati per quello scopo specifico.

8.2 Materiali Phong, GLSL e Point Sprite

I tre MAT più utilizzati sono:

- Il MAT "Phong";
- Il MAT "GLSL";
- Il MAT "Point Sprite".

I vari utilizzi di questi MAT saranno affrontati velocemente in questa sezione. Sono tutti e tre Operatori molto diversi fra loro, ma insieme soddisfano molte delle richieste di applicazione di shading e materiali.

Il MAT "Phong" è l'Operatore materiale più comune: è responsabile per l'applicazione delle texture alle geometrie 3D. Esistono una varietà di mappe che possono essere applicate, come "Color", "Bump", "Specular", "Diffuse" e altre. Il MAT "Phong" può essere utilizzato con luci "Ambient", Diffuse", "Specular", "Emit" e "Constant". Aprite l'esempio "Phong.toe". In questo progetto sono presenti due esempi molto semplici di utilizzo del MAT "Phong": il primo sfrutta il canale alfa della texture per creare un cubo trasparente; il secondo imposta la "Emit Light" a (1,1,1) per illuminare completamente l'oggetto, indipendentemente dalle condizioni di illuminazione. Nella sezione degli esempi verranno presentati molti altri esempi relativi ai MAT "Phong".

Il MAT "GLSL" è utilizzato per creare materiali personalizzati utilizzando l'OpenGL Shading Language (GLSL in breve). Il GLSL è un fantastico linguaggio di programmazione che può creare texture estremamente complesse in grado di essere eseguite molto velocemente. Riesce a fare ciò fornendo al programmatore un discreto controllo sulla pipeline grafica, senza però esporlo al linguaggio assembly. All'inizio ci si può scontrare con una curva di apprendimento piuttosto ripida, ma su Internet si possono trovare moltissimi esempi di shader GLSL, così come nell'area "Shared Components" del Forum di TouchDesigner, dove è presente un sostanzioso numero di ottimi modelli.

Il MAT "Point Sprite" è utilizzato per assegnare sprite ai punti di un sistema di particelle. Il nome è auto-esplicativo, dato che posiziona un'immagine 2D (uno sprite) in ogni punto dello spazio 3D. Gli sprite sono sempre orientati verso la telecamera e sono scalati in accordo con la loro distanza dalla telecamera. L'esempio "Point_Sprite.toe" dimostra tale comportamento. Per creare un Network simile in TouchDesigner senza il MAT "Point Sprite" si otterrebbe un progetto piuttosto disorganizzato, di chissà quanti TOP "Transform" e "Composite", e tutti questi Operatori utilizzerebbero molte più risorse. Sfruttando invece un sistema di particelle e il MAT "Point Sprite" il network risulta di facile lettura e non richiede molte risorse di sistema.

8.3 Mappe UV

Le mappe UV sono essenziali per lavorare con geometrie 3D complesse; come con certi aspetti della modellazione 3D, è più facile creare e lavorare queste mappe in un programma dedicato.

Le mappe UV sono ciò che permette a designer ed artisti di creare interessanti movimenti e texture grafiche fisse per le geometrie 3D; collegano il mondo 2D, in cui vengono creati video e immagini, con il mondo 3D delle geometrie.

L'applicazione delle mappe 3D è un processo in 3 passaggi: il primo passo è aprire l'oggetto 3D in un piano 2D. Questa texture aperta è chiamata mappa UV. Ci si riferisce ad essa come ad una mappa perché, in modo simile ad ogni altro tipo di mappa, prende un oggetto 3D e crea un corrispettivo 2D in modo accurato e proporzionale: è lo stesso processo che avviene nelle mappe stradali o del mondo, che prendono l'universo 3D e lo rappresentano su un piano 2D.

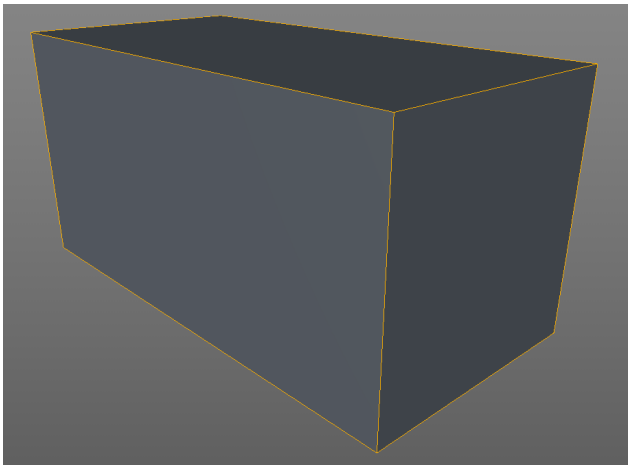
Il secondo passo consiste nell'applicare le texture in 2D. La mappa UV 2D è utilizzata da artisti e designer nei loro software di compositing per creare texture, fisse o in movimento. Il vantaggio di usare una mappa UV è che la texture può essere applicata alla geometria con un alto grado di precisione.

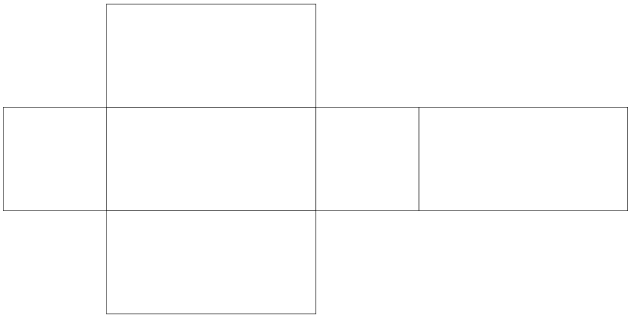
Il terzo ed ultimo passaggio è l'applicazione della texture sulla geometria 3D; questo processo varia a seconda del software utilizzato.

La combinazione di questi tre passaggi è chiamata UV mapping.

L'ultimo passaggio è un'operazione compiuta in modo relativamente frequente direttamente all'interno di TouchDesigner: se la geometria 3D è stata esportata correttamente dal suo software di modellazione, includerà delle coordinate che indicano alla altre applicazioni dove devono essere applicate le mappe UV; in queste situazioni la texture è caricata in un TOP "Movie File In" ed applicata sulla geometria utilizzando un MAT "Phong". Se è necessario apportare qualche modifica al modo in cui la mappa UV è stata applicata, è possibile utilizzare il SOP "Texture".

Di seguito è riportato un esempio di un semplice cubo 3D con la sua mappa UV:





9 Python

9.1 Introduzione

Lo scripting in Python è una delle caratteristiche più potenti di TouchDesigner: permette infatti di svolgere operazioni incredibilmente complesse, come iterare attraverso larghi insiemi di dati, comunicare nativamente con miriadi di API web, controllare e cambiare i parametri degli Operatori in modo estremamente rapido e molto altro.

È importante notare che, al momento in cui si scrive, TouchDesigner 088 utilizza Python 3.3.1. Python 3 si differenzia da Python 2.7 in molti modi, ma la maggior parte dei problemi può essere risolto con una rapida e indolore ricerca su Google. Molte domande sono probabilmente già state poste, e la soluzione potrebbe essere tanto semplice quanto spostare delle parentesi.

Per esempio in Python 2.7 la funzione "print" poteva essere utilizzata senza parentesi, mentre in Python 3 questo genera un errore. Di seguito è riportato un esempio di come viene utilizzata la funzione "print" in Python 2.7:

```
1  print 'text'
```

Questo è invece il modo in cui la stessa funzione è utilizzata in Python 3, dove sono richieste le parentesi:

```
1  print ('text')
```

Una caratteristica formidabile di Python, oltre alla sua affidabilità, è la possibilità di utilizzare numerose librerie esterne, molte delle quali permettono di connettersi nativamente alla maggior parte delle piattaforme più frequentate del web, come Facebook, Twitter, Instagram, Foursquare, eccetera. Questo tipo di integrazione nativa permette di progettare sistemi basati sulla raccolta dati in tempo reale, così come incredibili database per una visualizzazione e presentazione coinvolgente ai clienti.

Negli esempi seguenti sarà presente un po' di scripting in Python e, anche se non è obbligatorio, è altamente consigliato investire un po' di tempo per imparare Python seguendo qualche tutorial introduttivo. Esistono molte fantastiche risorse online, alcune persino sotto forma di giochi, e la maggior parte si prestano a lezioni mordi e fuggi. Spesso 10-15 minuti di tutorial al giorno nel corso di poche settimane possono fornire una buona dose di conoscenza delle basi di Python, e rivelarsi più che sufficienti per implementare gli script in TouchDesigner.

9.2 Textport

Il Textport svolge alcuni ruoli importante per lo scripting in TouchDesigner. Esistono due modi di aprire il Textport: il primo è utilizzare la combinazione di tasti "Alt + T"; il secondo è selezionare la voce "Textport" da "Dialogs" nella barra dei menu posizionata in alto nella finestra di TouchDesigner.

Il primo ruolo chiave del Textport è il suo utilizzo in modo simile alla shell IDLE compresa nelle installazioni di Python. Aprite il Textport e, come esempio, scrivete il comando seguente, premendo quindi "Invio" sulla tastiera:

```
print(2+2+2+2)
```

L'esecuzione di questo script in Python fa comparire il risultato dell'equazione, che è 8: dopo aver premuto "Invio" nel Textport viene mostrato il risultato cercato. Questo avviene perché il Textport funziona come un interprete di Python, perciò scrivendo al suo interno alcuni righe di codice Python, questo sarà processato e verrà restituito il risultato.

Analogamente può fare la stessa cosa per tscript, il linguaggio di programmazione che veniva utilizzato in TouchDesigner 077. Notate che quando aprite il Textport la prima cosa mostrata è:

```
python >>
```

Questo perché in TouchDesigner 088 l'interprete del Textport è impostato di default su Python; per lavorare invece con tscript occorre cambiare da Python a tscript la modalità del Textport. Ciò è possibile facendo click sul logo di Python nell'angolo in alto a sinistra del Textport. Una volta premuto esso cambierà nella lettera "T", e la linea successiva nel Textport sarà:

```
tscript ->
```

Ciò significa che il Textport è ora in modalità tscript. Come ulteriore conferma scrivete il seguente codice tscript nel Textport:

```
echo Hello!
```

In modo simile alla funzione "print" di Python, questo script in linguaggio tscript stamperà la scritta "Hello!" nel Textport.

Un altro importante utilizzo del Textport è come debugger Python. Aprite l'esempio "Textport_1.toe".

Questo è un progetto molto semplice che evidenzia quanto sia utile il Textport quando si utilizzano script in Python. Aprite il Textport, poi fate click sul pulsante "Good Python" in centro al Network. Così verrà eseguito uno script che restituirà nel Textport:

```
this will not error
```

Ora fate click sul pulsante “Bad Python”. Il Textport mostrerà qualcosa di molto diverso:

```
File "/project1/chopexec1", line 11
        Print(this will error)
                             ^
SyntaxError: Invalid syntax
```

È un errore di Python, avvenuto perché una parte del codice interpretato non è corretta. Imparare a leggere questi errori può velocizzare enormemente il debugging per gli script in Python più lunghi. Esaminiamo le differenti porzioni di questo codice di errore.

La prima riga indica esattamente dov’è situato l’errore all’interno del Network, e quale linea dello script sia considerata errata dall’interprete Python:

```
File "/project1/chopexec1", line 11
```

In questo esempio l’Operatore contenente l’errore si chiama “chopexec1”, posto all’interno del componente “project1”. L’interprete Python si interrompe alla riga 11.

Al di sotto di queste indicazioni il Textport riporta la linea contenente l’errore:

```
Print(this will error)
```

Più spesso di quanto si creda è possibile individuare errori di sintassi e di battitura in queste due righe. In questo caso è chiaro che il testo che dovrebbe venire restituito non è stato racchiuso fra virgolette. Sapere dov’è l’errore, e nello specifico in quale linea esso avvenga, significa poter risolvere il problema molto rapidamente. Osservate ora l’ultima riga dell’errore di esempio:

```
SyntaxError: Invalid syntax
```

Questo è il tipo di errore che Python ha incontrato. Informazioni più dettagliate sulle diverse tipologie di errori possono essere trovate consultando la documentazione ufficiale di Python 3.3, nella sezione “Errors and Exceptions”. Il “Syntax Error” è un errore molto comune, e compare quando l’interprete Python incontra del codice che non è stato scritto seguendo la sintassi corretta. Come già detto, nella riga considerata mancano le virgolette intorno alla scritta da far stampare.

Stampare nel Textport è un modo eccellente per rimuovere un po’ di mistero dallo scripting. Quando si scrive codice in progetti sempre più corposi, capita spesso di avere script all’interno di ogni Network, e il più delle volte molti di questi stanno eseguendo compiti nascosti, come caricare e rimuovere dalla memoria filmati, operazioni difficili da visualizzare. Inoltre moltissime volte più script funzionano in parallelo all’interno di Network differenti, e diventa incredibilmente difficile capire se il loro timing sia corretto, o se le azioni stiano accadendo nell’ordine sbagliato.

Utilizzando delle semplici funzioni “print” distribuite opportunamente negli script, non solo è facile vedere quando questi vengano eseguiti, ma è possibile ottenere un gran numero di informazioni: alcune di queste possono essere semplici, per esempio il percorso dello script eseguito, altre invece possono essere molto dettagliate, indicando i valori e le variabili con cui si sta lavorando e che sono stati modificati.

Aprite l’esempio “Textport_2.toe”. In questo progetto sono presenti due sequenze di script che vengono eseguite una dopo l’altra, con un certo intervallo tra di loro. Fate click sul pulsante “Visual Sequence”. Questa sequenza ha dei pulsanti che vengono spostati nella posizione “On” ogni volta che viene eseguita: servono puramente per rendere più facile visualizzare il progresso di questa sequenza, ma come si potrebbe controllare lo stesso progresso da un altro Network?

Aprite il Textport e premete il pulsante “Textport Sequence”. Al contrario della prima sequenza, questa stampa dei messaggi nel Textport per ogni script eseguito. Non ci sono pulsanti all’interno del Network che cambiano visivamente stato, ma abbiamo appena ottenuto accesso ad un gran numero di nuove possibilità: la prima è l’abilità di controllare l’andamento di questi script da qualsiasi posizione nel progetto; la seconda è che diventa possibile comparare il timing di queste sequenze ad ogni altro script che dovesse essere eseguito, anche in altre posizioni del Network; la terza, e probabilmente la più preziosa, è l’abilità di mettere in pausa il progetto e conservare una cronologia scritta della sequenza di eventi più recente.

La cronologia diventa assolutamente indispensabile quando si esegue il debug di sistemi logici complessi. In un progetto con 30 Network e 300 script indipendenti, se una serie di azioni si bloccasse senza restituire errori in Python sarebbe impossibile rintracciare i problemi senza un report ordinato degli eventi. Man mano che uno script diventa più lungo e complesso è facile creare sempre più di questi pseudo-checkpoint all’interno dello script, come per esempio indicazioni di cosa sta venendo calcolato, con messaggi simili a: “eseguo la sezione X dello script Y”.

9.3 Pratiche Consigliate

Esistono alcune pratiche consigliate che dovrebbero sempre essere tenute presenti quando si lavora con Python. Un elemento interessante riguardo questo linguaggio è il fatto che esso sia nato e cresciuto intorno alle idee di leggibilità e semplicità. Python contiene nascosto al suo interno un easter egg, un piccolo segreto che può essere rivelato digitando il seguente codice ne Textport o in un qualsiasi interprete Python:

```
import this
```

In questo modo Python restituisce una poesia chiamata “Lo Zen di Python”, di Tim Peters. Di seguito è riportato il testo:

```
The Zen of Python, by Tim Peters

Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
Special cases aren't special enough to break the rules.
Although practicality beats purity.
Errors should never pass silently.
Unless explicitly silenced.
In the face of ambiguity, refuse the temptation to guess.
There should be one-- and preferably only one --obvious way to do it.
Although that way may not be obvious at first unless you're Dutch.
Now is better than never.
Although never is often better than *right* now.
If the implementation is hard to explain, it's a bad idea.
If the implementation is easy to explain, it may be a good idea.
Namespaces are one honking great idea -- let's do more of those!
```

Questa poesia è diventata il motto di molti sviluppatori Python, e i suoi versi cercano di trasmettere gli ideali di Python e un insieme di idee comuni ad ogni bravo sviluppatore.

Pensate alla riga:

```
'Explicit is better than implicit'
```

Questo concetto può essere applicato ad uno degli elementi più spesso trascurato, la nomenclatura. In Python e in TouchDesigner sono presenti numerose aree diverse in cui alcuni oggetti fanno riferimento ad altri sfruttandone il nome. Uno sviluppatore deve inoltre assegnare un nome alle variabili, alle funzioni, agli Operatori, ai Network ed a molto altro. Senza attenzione nel rinominare gli Operatori sarebbe impossibile individuarne rapidamente la funzione; in modo simile, senza attenzione ai nomi delle variabili leggere gli script può diventare un compito piuttosto tedioso, come mostrato prima in questo capitolo. Esistono due convenzioni che sono regolarmente utilizzate quando si assegnano i nomi agli Operatori e alle variabili in TouchDesigner.

La prima riguarda l'utilizzo dei trattini bassi al posto degli spazi. Qui alcuni esempi:

- final_comp
- stop_button
- time_now

I trattini bassi rendono i nomi facili da leggere e veloci da capire. Ci sono persone a cui non piace utilizzare i trattini bassi, e perciò è nata la seconda convenzione, che consiste nell'utilizzare lettere maiuscole per differenziare le parole. Ecco degli esempi:

- finalComp
- stopButton
- timeNow

Entrambe le modalità riflettono l'idea originaria di utilizzare nomenclature esplicite, ed è fortemente consigliato farvi sempre ricorso per facilitare la collaborazione.

9.4 Commentare il Codice

Commentare il codice è una pratica che non dovrebbe essere trascurata dai neofiti di Python. Se scrivere poche righe di codice è piuttosto semplice, commentarle lo è ancora di più: occorrono solo pochi secondi più per documentare velocemente cosa queste righe facciano, e questo torna incredibilmente utile in una moltitudine di situazioni. Ad esempio per:

- Condividere porzioni di codice e progetti con altri programmatori;
- Aggiornare il codice in vecchi progetti;
- Cambiare le funzionalità all'interno di un componente riutilizzabile.

Può sembrare inutile quando gli script sono brevi, ma è un'abitudine che dovrebbe essere rinforzata fin dal primo istante. Analizziamo l'esempio seguente:

```
a = 10
b = op('value')[0]

op('op12').par.rx = b

c = str(op('numbers')[0])
d = 'testing'
e = 'something'

op('lines').par.text = d + e + c
```

Questo script è difficile da leggere per svariate ragioni: il motivo principale è che le sue azioni non appaiono ovvie quando lo si osserva velocemente. Un modo semplice di aumentare la leggibilità dello script è commentarlo. Prendiamo lo script precedente e aggiungiamogli alcuni rapidi commenti:

```
#Set initial variable
#get the value from slider input
a = 10
b = op('value')[0]

#assign the slider value to video input rotation
op('op12').par.rx = b

#take numeric input from computer keypad
c = str(op('numbers')[0])
d = 'You'
e = 'pressed'
```

```
13
14 #create a sentence from variables above
15 #add it to a Text TOP to display
16 op('lines').par.text = d + e + c
```

Senza investire tempo per creare nomi delle variabili e degli Operatori che siano significativi, lo script ora è comunque molto più facile da leggere e, anche se preso fuori dal contesto, la sua funzione è palese.

Queste semplici aggiunte possono rendere la collaborazione con altri sviluppatori molto più facile e piacevole.

9.5 Compartimentalizzare

Più un progetto si fa complicato e più gli script diventeranno lentamente sempre più lunghi. Ad un certo punto occorrerà più tempo per scorrere il codice che per cambiarlo o aggiungere sezioni. Perciò è importante compartimentalizzare tutti gli script di un progetto.

Questo implica varie cose, ma porta un gran numero di vantaggi che permetteranno di migliorare direttamente il flusso di lavoro, ma soprattutto che eviteranno molti problemi quando si lavora in team. Alcuni dei benefici includono:

- Manutenzione a lungo termine degli script più semplice;
- Meno tempo speso a spiegare gli script ai colleghi;
- Riutilizzabilità più semplice delle funzionalità programmate;
- Modifiche al codice più rapide.

Aprite l'esempio "Scripting_1.toe". In questo progetto sono presenti dieci TOP "Movie In", ognuno con una seria di azioni da eseguire: per prima cosa ognuno di essi deve liberare la memoria occupata; quindi ad ognuno deve essere assegnato un nuovo percorso; infine ogni filmato deve essere caricato nel TOP "Movie In" corrispondente e preparato a venire riprodotto. Dato che questo esempio riguarda Python, tutte le azioni saranno eseguite con uno script Python. Guardate il DAT "Text" chiamato "movies".

Una veloce nota riguardo questo script: per comodità di questo esempio il programma non itera su tutti gli Operatori utilizzando dei cicli, così da simulare come potrebbe apparire uno script più lungo e complesso, anche se vengono eseguite solo poche semplici funzioni.

Fate click con il tasto destro sul DAT chiamato "movies" e selezionate "Run", così tutte le azioni appena menzionate saranno eseguite in sequenza. Questo script non è molto commentato, perciò scorrerlo velocemente può disorientare un po'. Per modificare un singolo valore nascosto da qualche parte nel codice occorre individuarlo all'interno della lunga lista di azioni; inoltre eventuali colleghi impiegherebbero molto tempo per capire cosa fa questo script; infine, per riutilizzare una parte di questo script in futuro gli spezzoni andrebbero trovati ed estratti manualmente: come possono questi processi diventare più semplici?

Aprite l'esempio "Scripting_2.toe". Questo progetto prende il codice dell'esempio precedente e separa ogni "azione" in un DAT "Text" individuale. Immediatamente, anche senza addentrarsi a fondo nei vari script, è facile dire cosa faccia ognuno di essi: uno svuota le memorie occupate, uno cambia i percorsi assegnati e uno per carica i filmati desiderati. Alla fine di ognuno è presente una linea di codice che esegue quello successivo, utilizzando la funzione "Run":

```
op('preload').run()
```

Se volessimo modificare velocemente un singolo valore sarebbe facile, poiché le azioni e i parametri sono più rapidi da rintracciare; passare questo componente ad un collega non sarebbe affatto un problema, dato che potrebbe capire subito cosa fa ogni singolo script; se infine qualcuno avesse

bisogno di uno script per caricare un insieme di TOP “Movie In”, sarebbe immediato estrarlo da questo progetto.

Questo flusso di lavoro compartimentalizzato aiuta a spezzare gli script per gestirli meglio, prendendone di lunghi, a volte con più di 500 righe, e trasformandoli in script più piccoli, facili da consultare e condividere. Nel caso dell’esempio precedente, esiste un terzo metodo per gestire il lavoro con più Operatori.

Aprite l’esempio “Scripting_3.toe”. Questo progetto si avvantaggia delle funzioni di Python, piccole porzioni di codice che possono essere richiamate per eseguire una serie di azioni. All’interno del DAT “Text” chiamato “actions”, è stata definita una funzione contenente l’insieme di azioni che devono essere eseguite su ogni TOP “Movie In”. Dal DAT “Text” chiamato “set_movie_tops”, al posto di riscrivere lo stesso insieme di azione più e più volte, viene richiamata la funzione, fornendole semplicemente il nome di ogni TOP “Movie In”.

Anche se queste azioni sono abbastanza arbitrarie, l’idea è semplice: è sempre utile compartimentalizzare gli script Python per avere manutenzione semplificata, lavori collaborativi più facili, riutilizzabilità e gestione più comoda.

9.6 Moduli Esterni

Uno dei vantaggi meno sfruttati nell'integrazione di Python in TouchDesigner è la possibilità di importare librerie di terze parti e di utilizzarle nativamente, come le popolari "Requests" o "Beautiful Soup". Importare una libreria è un processo semplice:

1. Installate una build a 64-bit di Python. Al momento della scrittura, la più recente è la 3.5;
2. Usate "pip" per installare i pacchetti desiderati;
3. Aprite TouchDesigner, andate nel menu "Edit" e fate click su "Preferences";
4. Nella sezione "General" cercate l'impostazione chiamata "Python 64-bit Module Path" e aggiungete il percorso alla cartella "site-packages" del vostro Python a 64-bit. Questa cartella può essere trovata nella cartella di installazione di Python, dentro alla cartella "Lib";
5. All'interno di TouchDesigner create un DAT "Text" e testate il pacchetto utilizzando il comando standard "import".

Una buona idea è confermare che "pip" abbia installato il pacchetto per la versione corretta di Python: un problema più frequente di quanto si possa immaginare è che "pip" venga utilizzato per una versione di Python diversa da quella desiderata.

9.7 Dove Imparare Python

Di seguito è riportata una lista molto basilare di alcuni tutorial online gratuiti di Python (Nota: alcuni dei tutorial sono per Python 2.7, ma sono fatti incredibilmente bene, e perciò inclusi).

CodeAcademy http://www.codecademy.com/tracks/python[4]

Khan Academy https://www.khanacademy.org/science/computer-science[5]

Coursera https://www.coursera.org/course/interactivepython[6]

Google Developers https://developers.google.com/edu/python/?csw=1[7]

Learn Python The Hard Way http://learnpythonthehardway.org/[8]

[4] http://www.codecademy.com/tracks/python
[5] https://www.khanacademy.org/science/computer-science
[6] https://www.coursera.org/course/interactivepython
[7] https://developers.google.com/edu/python/?csw=1
[8] http://learnpythonthehardway.org/

10 Mostrare i Contenuti per Installazioni e Performance

10.1 Modalità Perform

Quando si mostra un progetto finito bisognerebbe utilizzare sempre la modalità “Perform”, o almeno ogni volta in cui è possibile. La premessa fondamentale dietro all’utilizzo della modalità “Perform” è che, quando un progetto è pronto per essere consegnato, non dovrebbe richiedere altra programmazione, e perciò non necessiterà dell’editor del Network. È sorprendente quante risorse di sistema possano essere spese anche solo per renderizzare questo editor, specialmente se sono attivi molti viewer degli Operatori, che mostrano canali dei CHOP, anteprime dei TOP, tabelle nei DAT, eccetera.

Proprio per questo motivo esiste la modalità Perform: per far sì il computer possa concentrarsi sulla presentazione dei contenuti e non debba renderizzare i vari editor dei Network. Tutto ciò che è coinvolto nella creazione del prodotto finale viene comunque renderizzato e resta attivo nella finestra “Perform”, inclusi eventuali input e output di dati esterni, l’unica cosa che smette di essere renderizzata è l’editor del Network.

Come già detto nel capitolo sui Componenti, è raccomandabile utilizzare come sorgenti dei COMP “Window” i COMP “Container”; lo stesso consiglio è valido per la modalità “Perform”.

Aprite l’esempio “Perform_mode.toe”. In questo progetto il COMP “Container” chiamato “container1” è utilizzato come sorgente dei contenuti per la modalità “Perform”. Premete il tasto “F1” per avviare la modalità “Perform” e vedrete il contenuto di questo Container aprirsi in una nuova finestra; premete il tasto “ESC” per ritornare all’editor del Network. Esiste anche un pulsante nell’interfaccia utente che può essere utilizzato per entrare nella modalità “Perform”:

10.1.1

10.2 Performance con l'Editor del Network

Ogni situazione è diversa dalle altre, ed è possibile che sia necessario programmare durante una performance live. Se questo è il caso, esistono alcune pratiche che possono aiutare a minimizzare l'impatto dell'editor del Network sul sistema. La prima è disattivare i visualizzatori di tutti gli Operatori che non hanno necessità di essere monitorati. Osservate l'immagine seguente:

10.2.1

10.2.2

Un'altro consiglio è utilizzare un solo pannello nell'editor del Network. Più pannelli sono infatti visibili e più cose dovranno essere renderizzate, occupando risorse di sistema.

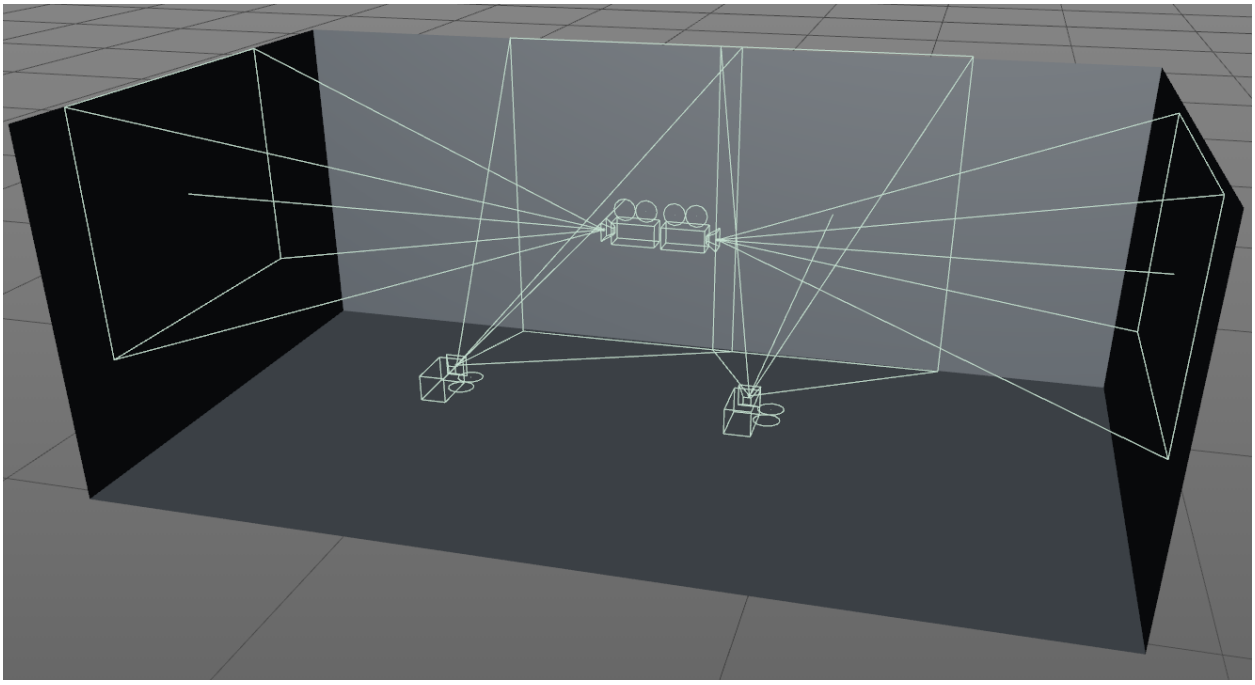
Infine bisognerebbe evitare di muoversi dentro e fuori da grandi Network. Spostarsi in questo modo può causare perdite di frame, dato che TouchDesigner all'improvviso si troverà a dover renderizzare tutti gli Operatori in quel Network prima di poterli mostrare.

10.3 Creare un Raster di Uscite

Come regola generale, per avere la miglior performance possibile bisognerebbe sempre cercare di utilizzare un solo COMP “Window” alla volta. Ovviamente questo non vale durante la fase di programmazione, ma solo quando occorre mostrare i contenuti per delle installazioni oppure durante le performance: in questi casi avere più di una finestra aperta riduce notevolmente le performance del sistema. Cosa occorre quindi fare quando è necessario gestire più output, magari posizionati e ruotati in maniere differenti?

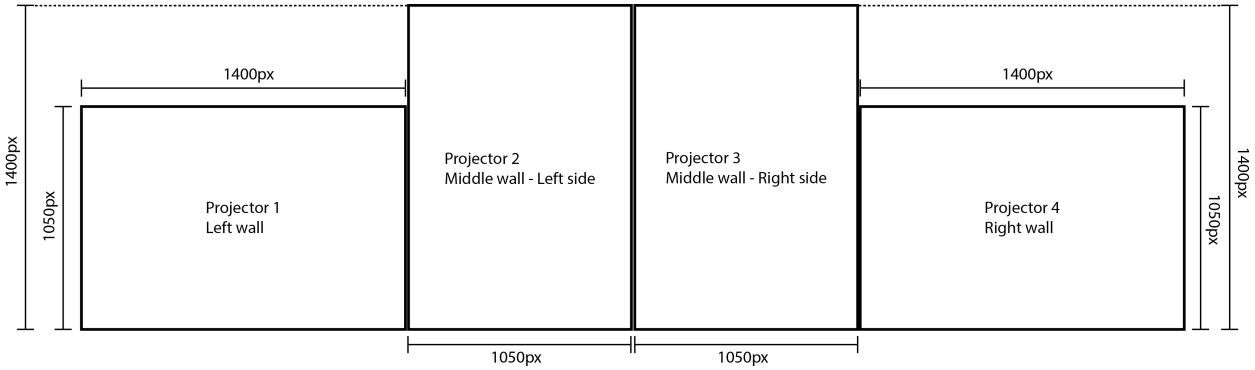
La risposta è creare un raster di output con una singola finestra che comprenda tutte le uscite.

È più immediato ricorrere ad un esempio. Consideriamo uno scenario che comprenda 4 proiettori SXGA+, ognuno con una risoluzione di 1400x1050 pixel. Nel setup reale sono presenti 2 proiettori puntati orizzontalmente sui muri laterali e due proiettori affiancati che puntano verticalmente sul muro centrale. L’immagine di seguito schematizza il setup cercato:



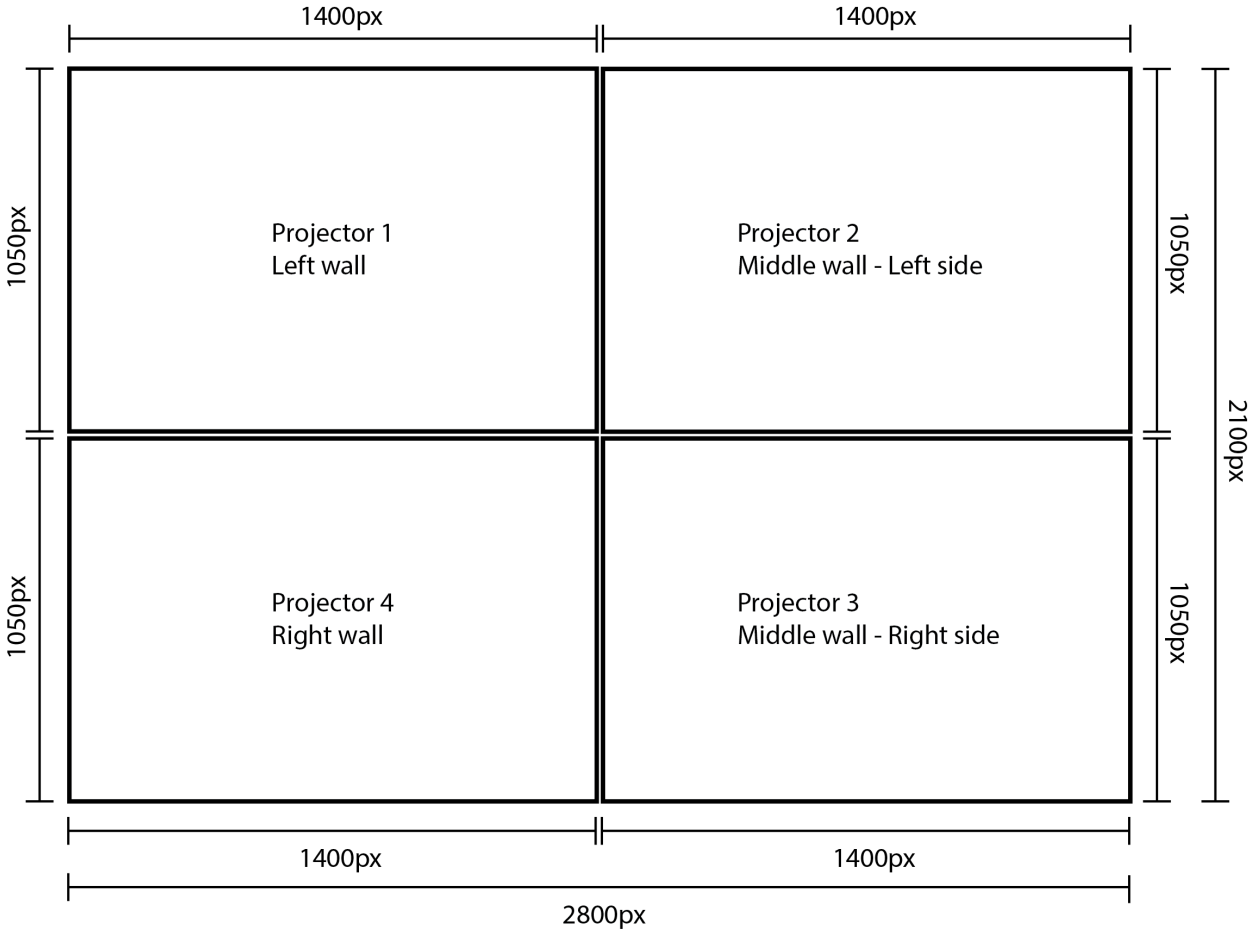
10.3.1

Questo setup non è particolarmente complicato, ma è importante saperlo gestire nel modo più appropriato. Prendiamo questo esempio e schematizziamolo in 2D:



10.3.2

L'approccio intuitivo di un principiante potrebbe essere quello di utilizzare 4 COMP "Window", visto che sono presenti 4 uscite, due delle quali devono essere ruotate. La sfida è trovare il layout più efficace per questi 4 riquadri, così da creare creare un singolo raster. In questo caso, poiché tutte le uscite sono alla stessa risoluzione, una soluzione semplice è creare una griglia 2x2:



10.3.3

Nell'immagine precedente tutte le uscite sono combinate in un unico raster. Questo setup riesce

quindi ad utilizzare in maniera efficiente un COMP "Window" da 2800x1200 pixel. A questo punto occorre usare il pannello di controlla nVidia o AMD per creare un setup simile per i monitor in Windows, da connettere poi alle correte uscite della scheda grafica.

Il passo successivo è preparare questo raster all'interno di TouchDesigner. Aprite l'esempio "Raster.toe". Notiamo un po' di cose fin dall'inizio: è stato creato del contenuto molto semplice come segnaposto, che rappresenterà dove andrà i media veri e propri. "Content1" è per il muro di sinistra, ed è da 1400x1050 pixel. "Content2" è invece per il set di proiettori centrali, e misura 2100x1400 pixel. Tutta la costruzione delle cornici avviene all'interno del COMP "Container" chiamato "canvas".

In questo "Container" è possibile seguire il flusso del segnale da sinistra verso destra, e dall'alto verso il basso, come una cascata. Il primo passo è creare un raster vuoto su cui sia possibile compositare il contenuto: per fare ciò è presente un TOP "Constant" impostato a 2800x2100 pixel, in alto a sinistra nel Network. Utilizzando un TOP "Over" si posiziona il primo frammento di contenuto, quello per il proiettore sul muro sinistro, nella posizione adatta, secondo lo schema precedente, utilizzando il parametro "Translate" nel TOP "over". Proiettore 1: completato!

Il muro centrale ha due proiettori con i bordi fusi insieme. Poiché diverse operazioni devono avvenire sul contenuto, è presente un "Container" chiamato "crop", che mantiene bene organizzate le operazioni, in modo pulito e facile da trovare. Dentro a "crop" vengono svolte tre operazioni principali: la prima è che la grossa porzione di contenuto è tagliata a metà, così che ogni proiettore possa mostrare metà dell'immagine. Poiché i proiettori sono posizionati verticalmente nell'installazione, ma sono posizionati orizzontalmente nel raster, è stato utilizzato il parametro "Flop" all'interno del TOP "Flip" per ruotare la cornice sul lato. Le impostazioni del TOP "Flip" finiranno per cambiare di volta in volta, a seconda del setup hardware, quindi siate pronti a provare diverse impostazioni di Flip e Flop per ottenere l'orientazione corretta.

Nota a margine: molti principianti hanno difficoltà a ruotare per intero una cornice. Il primo istinto è quello di utilizzare il TOP "Transform", ma è importante notare che il TOP "Transform" agisce sui pixel all'interno della cornice. Qui è invece dove torna utile il parametro "Flop" del TOP "Flip", permettendo di ruotare l'intera cornice.

Poiché questo esempio non è dedicato alla fusione dei bordi, i "Container" chiamati "edge_blend" sono solo dei segnaposto che creano l'effetto visivo di due bordi sovrapposti.

Con tutti i ritagli, le rotazioni e le fusioni compiute, i due output dei proiettori sono pronti per essere compositati sul raster: usando la stessa tecnica di prima, un TOP "Over" con il parametro "Translate" opportunamente modificato posizionerà in maniera corretta i due frammenti di contenuto. Ora sono completi anche i proiettori 2 e 3!

L'ultimo proiettore è semplice quanto il primo, e si può posizionare l'ultimo pezzo del puzzle utilizzando il fidato TOP "Over".

Come già detto in un capitolo precedente, è sempre meglio utilizzare COMP "Container" al posto di TOP come sorgenti per i COMP "Window". In questo progetto è perciò presente un "Container" da 2800x2100 pixel che contiene il raster completo. Il contenitore "final" è impostato come "Operator" del COMP "Window", l'impostazione "Borders" è spenta nel COMP "Window" e la dimensione della

finestra è impostata a 2800x2100 pixel: così il progetto è pronto per essere inviato al setup di 4 proiettori precedentemente descritto.

10.4 Display, Tearing, e Stuttering

All'interno di questo libro non affrontiamo molte discussioni riguardo l'hardware, ma quando si lavora con più display è doveroso ricordare alcune cose. La regola più importante è cercare di utilizzare sempre display tutti esattamente uguali; ogni differenza nei segnali può altrimenti causare il cosiddetto "tearing".

Osservate l'immagine seguente: è un esempio di come appare un fotogramma con del tearing, in cui potete facilmente notare i due tagli orizzontali che attraversano il frame.

Image courtesy of Wikipedia

Il fenomeno del tearing si manifesta quando un display aggiorna il suo contenuto senza essere sincronizzato con il rendering compiuto dalla scheda grafica. Il risultato è che una parte dell'immagine appartiene ad un frame, e una parte a quello successivo. Su immagini che si muovono lentamente a volte può essere difficile da notare, ma ogni qual volta sono presente dei movimenti piuttosto rapidi il tearing diventa una distrazione incredibile.

Il tearing è un problema complesso da risolvere, ma esistono alcune misure preventive che possono essere adottate per evitarne l'insorgere. La prima è di utilizzare una scheda grafica professionale, come quelle della seria Quadro di nVidia; la maggior parte delle compagnie non garantiscono infatti riproduzioni prive di tearing su schede diverse da quelle professionali.

La seconda è di assicurarsi sempre che i display siano identici, dove il concetto di identici non può mai essere sottolineato a sufficienza. Se consideriamo un'installazione con tre uscite con frequenza di aggiornamento di 60Hz e un'uscita con frequenza di 59Hz, esiste una possibilità concreta che si verifichi del tearing. Anche se consideriamo un'installazione con due proiettori 1080p e due proiettori SXGA+ è probabile che si manifesti del tearing. La pratica più consigliata è utilizzare display che siano identici non solo a livello di specifiche, ma che siano esattamente lo stesso modello. Un'altra causa del manifestarsi di tearing può essere l'utilizzo di applicazioni di accesso remoto, come VNC e LogMeIn.

Questo solleva un problema molto importante: con il tearing non esistono regole semplici e stringenti. A volte dei setup con una serie di risoluzioni e frequenze di aggiornamento diverse lavoreranno perfettamente e senza tearing; al contrario, altre volte persino setup con monitor identici possono manifestarlo. Non possiamo sottolineare a sufficienza l'importanza di prendere precauzioni per evitare la comparsa di tearing. Quando si manifestano problemi con il tearing, l'unica cosa che può essere fatta è un controllo e un'analisi passo passo di tutto il sistema, per verificare quale sia la causa scatenante.

Esiste una lista di metodi comuni di debug sulla pagina "Tearing" della Wiki di Derivative.

Generalmente i passi con cui iniziare sono i seguenti, in nessun ordine particolare:

- Verificare che il progetto non stia perdendo frame. La perdita di fotogrammi a volte può infatti innescare il tearing;
- Verificare che non siano attive altre applicazioni che stiano interrompendo o deviando i driver della scheda grafica, come ad esempio VNC e LogMeIn;
- Disconnettere tutti i display connessi al computer e riconnetterli individualmente finché non ricompare il tearing. Dopodiché isolare quel singolo display e capire se il problema sia dato da quello o dal layout utilizzato;
- Verificare nel pannello di controllo nVidia o AMD che tutti i display siano impostati alle stesse risoluzioni, profondità di colore e frequenze di aggiornamento;
- Verificare che nessun display abbia delle rotazioni applicate da Windows, poiché queste possono provocare comportamenti imprevisti;
- Controllare la versione dei driver della scheda grafica e aggiornarli se necessario;
- Controllare i driver e gli aggiornamenti firmware su eventuali video splitter esterni, come Datapath X4 o Matrox Triplehead2Go;
- Verificare di star renderizzando un unico COMP "Window";
- Verificare che Windows Aero sia disattivato. Su Windows 7, Aero può causare perdite di frame e stutter, ma evita il tearing. Una volta disattivato, il sistema può presentare tearing, ma è garantita una riproduzione senza stutter;

- Configurare un Mosaico Premium utilizzando il pannello di controllo nVidia per creare un singolo display logico.

Esistono molti casi in cui un sistema che si comporta perfettamente, e che non ha perdite di frame, avrà degli stutter occasionali. Questo può accadere quando i display e le schede grafiche negoziano frequenze di aggiornamento diverse.

Se un progetto gira a 30 FPS, ma la frequenza di aggiornamento dello schermo è 60 FPS, da qualche parte sarà necessario raddoppiare i frame. Il più delle volte questa negoziazione tra scheda grafica e display avviene in modo neutro, ma a volte possono verificarsi dei problemi: quel che può accadere è che, invece di compiere un corretto raddoppio dei frame, un fotogramma venga mostrato una volta e quello successivo tre volte. Dal punto di vista del progetto non vengono persi né tempo né fotogrammi, quindi non vengono mostrati problemi nel Performance Monitor o nel Task Manager di Windows.

Se sembra che possa verificarsi un problema di questo tipo si può sfruttare l'impostazione "Half Monitor Refresh" nel COMP "Window". Questo informa i driver della scheda grafica di mostrare un frame ogni due aggiornamenti.

10.5 Edge Blending

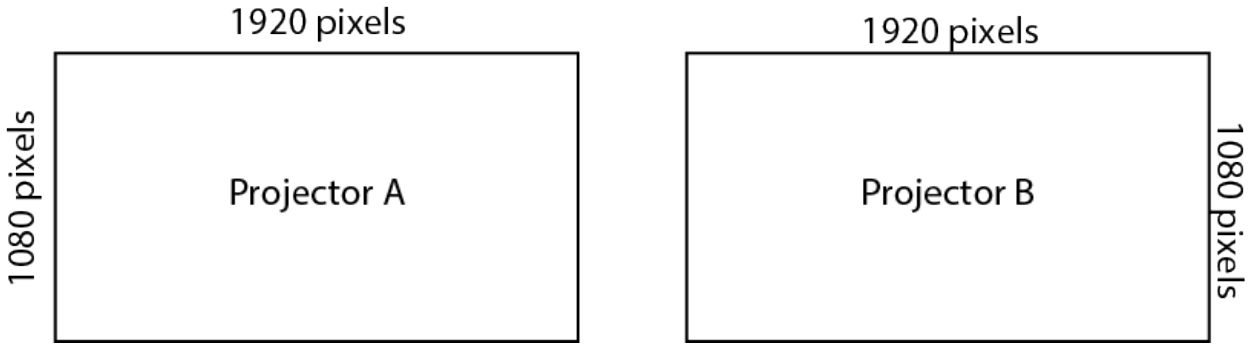
I videoproiettori offrono un'incredibile flessibilità e sono in grado di creare proiezioni infinitamente larghe: ciò è possibile creando array di proiettori con bordi sovrapposti, fusi insieme per creare un'unica tela. Questo atto di unire le sezioni sovrapposte è chiamato "Edge Blending".

L'idea dell'edge blending può essere nuova per chi generalmente lavora con insiemi di monitor e schermi. Un ottimo esempio dei principi di base celati dietro l'edge blending può essere presentato relativamente in fretta. Preparate due proiettori e fate sì che uno proietti il colore blu e l'altro il colore verde. Separatamente questi proiettori mostrano i colori a loro assegnati ma, se puntate i proiettori in modo che le loro proiezioni si sovrappongono, nell'area in cui i fasci si uniscono otterrete un color turchese. Questi principi di fisica e di sintesi additiva dei colori costituiscono le basi dell'edge blending.

C'è un eccellente articolo scritto da Paul Bourke che scende molto più in profondità nell'argomento di quanto sarà fatto qui. Per riferimenti e letture ulteriori trovate l'articolo al link seguente:

Edge blending using commodity projectors by Paul Bourke http://paulbourke.net/texture_colour/edgeblend/[9]

Il modo migliore per imparare le basi dell'edge blending è utilizzare un setup di esempio. In questo caso lo scopo è unire due proiettori da 1920x1080 pixel in un array 2x1 (una unità verticale e due orizzontali). Di seguito è riportato uno schema di un simile array 2x1 di proiettori:



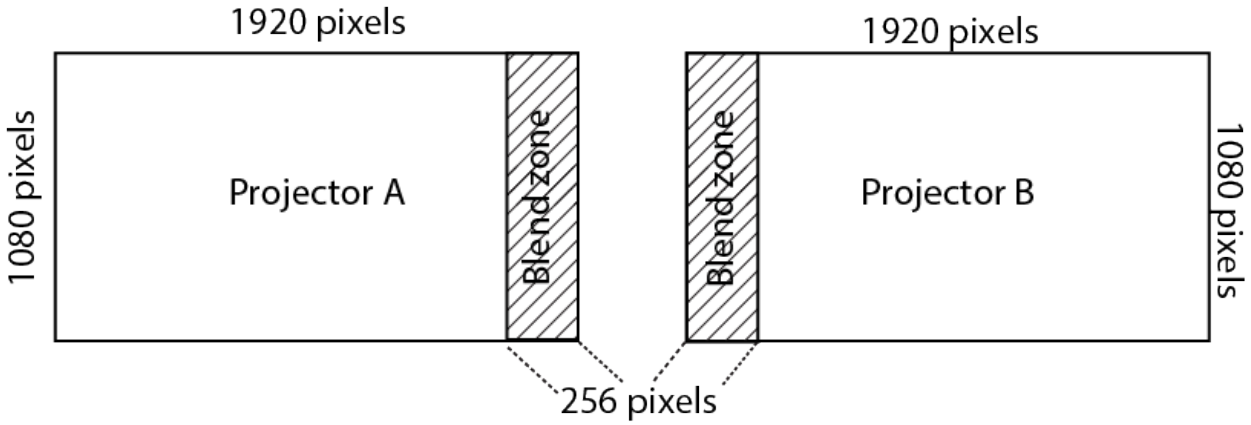
10.5.1

L'unione di entrambi questi proiettori richiederà la creazione di una sezione in cui le proiezioni si sovrappongano. L'ampiezza dell'area di sovrapposizione richiesta varierà in base a molti fattori, come la risoluzione dei proiettori e i parametri dell'installazione. Una buon inizio per sperimentare può essere utilizzare una zona che sia ampia quanto una potenza di 2 vicina al 10% della larghezza di una singola proiezione. Per questo esempio, il 10% della larghezza di uno schermo da 1920x1080 pixel è 192 pixel, e la potenza di 2 più vicina è 256.

Questa area di sovrapposizione può causare problemi se non vengono opportunamente considerate tutte le diverse implicazioni. Perché le immagini si sovrappongano, entrambi i proiettori devono

[9] http://paulbourke.net/texture_colour/edgeblend/

mostrare lo stesso contenuto nella sezione in cui si sovrappongono. In questo esempio ciò significa che il bordo destro del proiettore A ed il bordo sinistro del proiettore B devono avere esattamente lo stesso contenuto. Usando qualche numero ciò significa che i 256 pixel sul bordo destro del proiettore A devono essere uguali ai 256 pixel sul bordo sinistro del proiettore B, come nello schema di seguito:

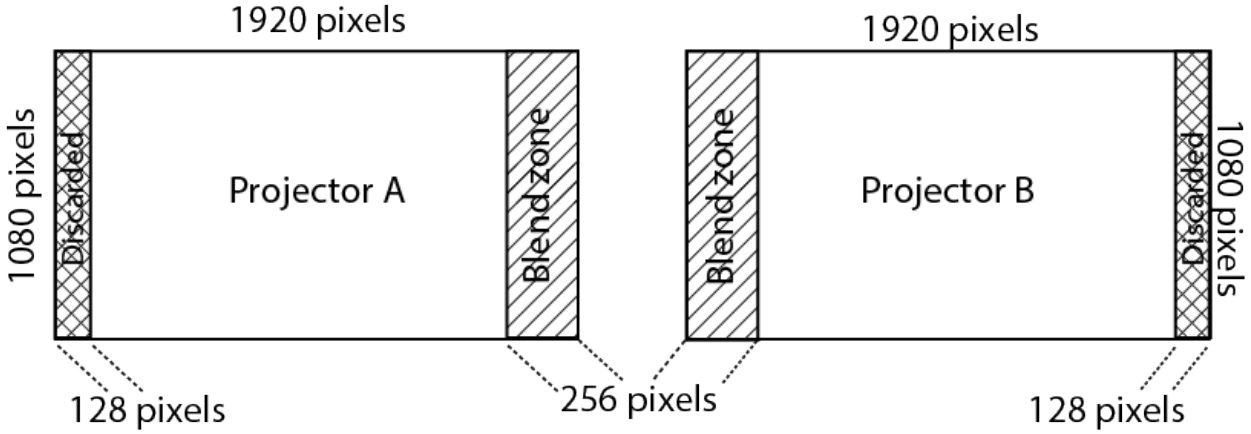


10.5.2

È importante tenere a mente questo fatto durante l'intero processo di produzione e per organizzare il flusso di lavoro. All'inizio può sembrare naturale pensare che un array 1x2 di proiettori da 1920x1080 pixel richieda la creazione di contenuti da 3840x1080 pixel, ma questo può portare a conseguenze indesiderate.

Un proiettore ha una quantità predefinita di pixel fisici, proprio come un monitor. Due immagini da 1920x1080 pixel posizionate fianco a fianco avranno effettivamente un'estensione di 3840 pixel ma, se è necessaria una sovrapposizione nella zona di fusione, i pixel extra utilizzati per l'edge blending non appariranno dal nulla. La zona di sovrapposizione avrà bisogno di consumare pixel che sarebbero stati altrimenti utilizzati per il contenuto.

In questo esempio se il team di creazione del contenuto avesse creato un asset da 3840x1080 pixel, 256 pixel andrebbero ora eliminati da qualche parte, per tener conto dei pixel raddoppiati nella zona di sovrapposizione. Da dove vengano eliminati questi pixel dipende dalle preferenze e dagli strumenti utilizzati, ma il metodo più comune è scartare metà della zona di sovrapposizione da ognuno dei bordi opposti alla zona di sovrapposizione. Questo metodo mantiene centrata la zona dove avviene la fusione, come nello schema di seguito:



Discarded Edge Layout

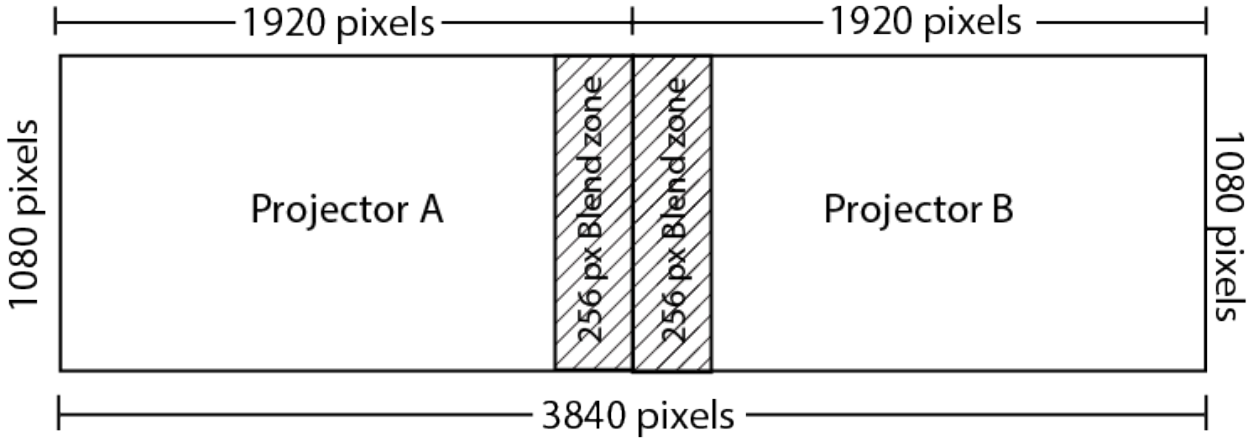
Quando si usano più proiettori occorre essere sempre coscienti di questa perdita di pixel, per evitare di posizionare contenuti o informazioni critiche in aree che rischiano di dover essere scartate. Perché però rimuovere solo metà della zona di sovrapposizione (128 pixel) da ogni lato, e non rimuovere l'intera area da entrambi i lati (256 pixel)? La risposta è che per creare una sovrapposizione di 256 pixel in centro, ogni lato deve essere avvicinato verso l'altro di 128 pixel. In questo modo la sovrapposizione risultante è di 256 pixel.

Prima di procedere con l'edge blending in TouchDesigner, riportiamo alcuni metodi di approccio alla creazione di contenuti per più proiettori da unire.

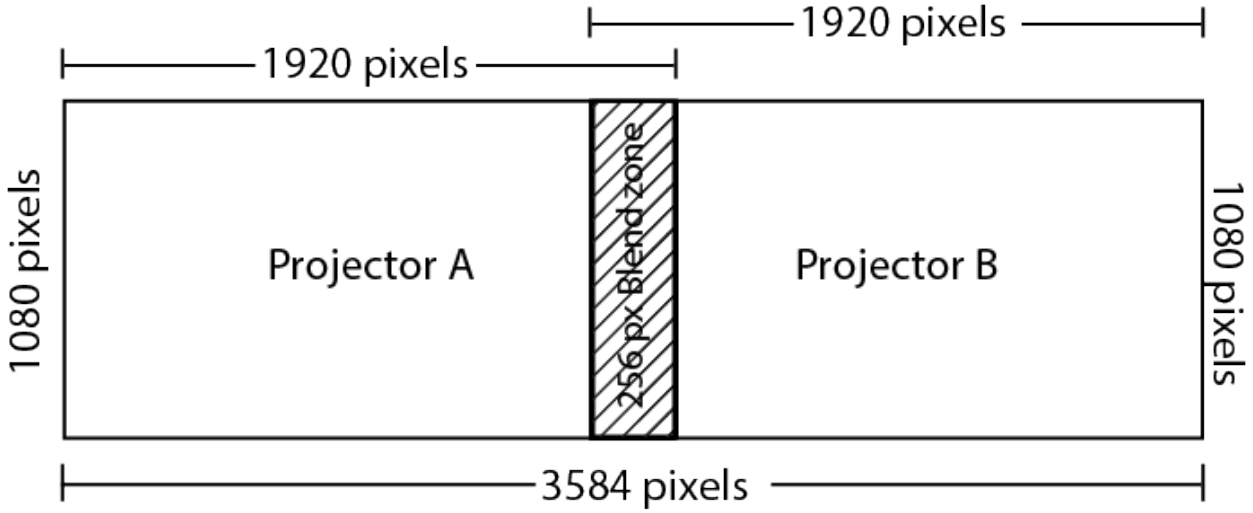
La prima modalità consiste nel creare contenuti sfruttando la piena risoluzione dell'uscita video, in questo caso 3840x1080 pixel, con una zona vicina ai bordi dove i pixel saranno scartati. Anche se una parte di questa zona verrà scartata, si può avere una maggior flessibilità per eventuali cambi di dimensione durante l'installazione e il setup, senza preoccuparsi di tagliare troppo contenuto o di non averne abbastanza per riempire la proiezione. Questo è il metodo prudente di procedere, per progetti che non hanno un proiezionista esperto nel loro staff che calcoli la zona di sovrapposizione prima dell'installazione.

Per progetti che coinvolgono un proiezionista o un'altra figura con esperienza nel team di lavoro, esiste l'opzione di creare contenuti della stessa dimensione dell'immagine visibile, e non della piena risoluzione dell'output. Questo è spesso comodo perché significa avere meno pixel da processare in flussi di lavoro generativi, o meno pixel da leggere da un hard disk. Per calcolare la dimensione totale dell'immagine in questo esempio prendete la risoluzione totale in uscita (3840x1080 pixel) e sottraete la dimensione della zona di fusione (256 pixel). Questo lascia 3584x1080 pixel per il contenuto o per le immagini. Con questo metodo è presente tuttavia il rischio che, se le zone di fusione non fossero state calcolate correttamente, il contenuto non risulti largo a sufficienza per riempire lo schermo senza ricorrere a riscalature successive.

Qualsiasi strada si scelga, lo schema seguente mostra nella prima immagine l'uscita finale di ogni proiettore uno di fianco all'altro, come vista da TouchDesigner e da Windows, e nella seconda l'immagine proiettata come vista sulla superficie di proiezione dopo l'edge blending.



Full Layout



Full Image Layout

Implementare in TouchDesigner questo esempio è immediato, grazie agli strumenti di edge blending. Prima di utilizzare il componente precostruito osserviamo gli elementi di base di un tipico processo di edge blending con un semplice esempio.

Aprite l'esempio "Simple_blend.toe". In questo progetto il COMP "Container" chiamato "content" crea del contenuto da 3840x1080 pixel deformando un file campione di TouchDesigner. L'immagine viene quindi divisa in due parti sfruttando dei TOP "Crop", uno per ogni proiettore. La cosa importante da notare è che lo spostamento avviene all'interno dei TOP "Crop". Il TOP "Crop" relativo al proiettore A crea una texture di 1920x1080 pixel (uguale alla risoluzione del proiettore) che parte dal bordo sinistro spostato di 128 pixel a destra, scartando quindi 128 pixel (metà della zona di sovrapposizione) dal bordo non fuso, in questo caso proprio il sinistro. Analogamente, il proiettore B crea una texture di 1920x1080 pixel che parte dal bordo destro spostato di 128 pixel a sinistra, scartando perciò 128 pixel dal bordo non fuso, che in questo caso è il destro. Avendo spostato entrambe le texture di 128 pixel una verso l'altra, si viene a creare una sovrapposizione di 256 pixel che servirà per l'edge blending.

Per creare la zona di sovrapposizione, tali texture devono essere moltiplicate per delle rampe della stessa dimensione della zona di fusione, che ricordiamo misura 256x1080 pixel. Sarebbero richiesti un certo numero di altri passaggi per creare un'immagine perfetta, ma questo esempio si limita a dimostrare il flusso di lavoro base. A questo punto gli elementi così preparati per essere fusi possono essere appropriatamente compositati su una singola texture alla piena risoluzione dell'uscita oppure, come in questo caso, le texture individuali possono essere assegnate allo sfondo dei COMP "Container", alla stessa risoluzione del proiettore. Questi COMP "Container" sono poi imparentati e allineati utilizzando il parametro "Align Layout Horizontal Left To Right" del "Container" genitore, che costituisce l'effettiva uscita finale.

In questo questo esempio basilare mancano alcuni elementi, come la correzione della gamma e della luminosità, ma il progetto serve a dare un'idea dei processi coinvolti: il ritaglio della texture iniziale, lo spostamento delle sezioni per creare la sovrapposizione, la creazione della zona di fusione utilizzando una rampa, e la creazione di un'immagine finale. Ora che abbiamo una comprensione basilare di questo flusso di lavoro, osserviamo un altro esempio utilizzando lo strumento già presente nel software, molto più ricco di funzioni.

Aprite l'esempio "Full_blend.toe". Questo progetto contiene due esempi corrispondenti ai due metodi di creazione del contenuto che sono stati appena discussi. Per trovare il componente di edge blending aprite il Browser della Palette usando la combinazione di tasti "Alt+L", oppure andando nel menu "Dialog" in alto nello schermo e facendo click su "Palette Browser". All'interno del Browser della Palette, nella sezione "Tools", troverete il componente "EdgeBlend" che può essere trascinato nel vostro progetto. Questo componente si basa sullo stesso articolo di Paul Bourke citato in precedenza.

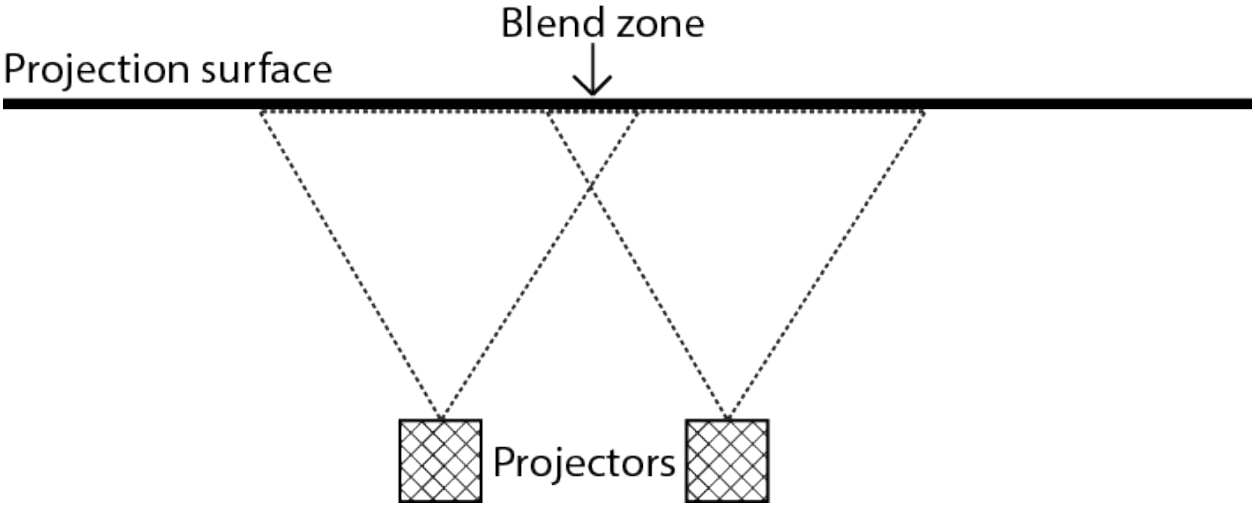
In entrambi gli esempi il componente "EdgeBlend" è impostato per avere un'area di sovrapposizione larga 256 pixel. Il componente "EdgeBlend" crea una sovrapposizione del contenuto al centro della texture, scartando dei pixel dai bordi non fusi. Esso moltiplica i bordi sovrapposti per una rampa e composita le texture così processate su una cornice che è impostata alla piena risoluzione del proiettore.

Il primo esempio utilizza del contenuto alla piena risoluzione dell'uscita, 3840x1080 pixel. Questo è il metodo prudente menzionato in precedenza, in cui perdere del contenuto sui bordi non fusi permette un po' più di flessibilità nella calibrazione della zona di sovrapposizione durante la fase di installazione.

Il secondo esempio usa contenuto alla risoluzione di 3584x1080 pixel. In questo modo non è necessario scartare pixel da questa texture. Una cosa fondamentale da ricordare è che molti strumenti di fusione dei bordi daranno per scontato che si stia utilizzando la strada del primo esempio, in cui si fornisce contenuto a piena risoluzione e in cui occorra scartare pixel dai bordi esterni, restituendo una texture alla stessa risoluzione di quella iniziale. Questo è il motivo per cui in questo esempio l'asset della risoluzione dell'immagine (3584x1080) è compositata nel centro di una cornice a piena risoluzione (3840x1080) prima di essere inserita nel componente "EdgeBlend": così il componente "EdgeBlend" raddoppierà il contenuto centrale e scarterà lo spazio vuoto situato su entrambi i lati dell'immagine.

La differenza delle uscite può essere vista chiaramente, dato che l'esempio più in alto perde la barre colorate che erano sui bordi dell'immagine, mentre il secondo esempio le mantiene.

Con le uscite così preparate è solo questione di mandare la texture corretta al proiettore corretto. Come collegare i proiettori e lavorare con setup complessi è al di fuori dello scopo di questo capitolo. In una situazione ideale, entrambi i proiettori sono paralleli alla superficie di proiezione come mostrato:



Projectors

Se la quantità di pixel nella zona di sovrapposizione è già conosciuta, inseritela nel parametro "Region" del componente "EdgeBlend", quindi provate a disattivare il pulsante "Enable Blend" per essere sicuri che le vostre immagini si sovrappongano correttamente. A questo punto le immagini dovrebbero corrispondere ed essere allineate. Ricordate sempre che lo scopo dell'edge blending è di rimuovere le imperfezioni, non di allineare il contenuto.

Se la quantità di pixel nella zona di sovrapposizione non è invece nota, disattivate il pulsante "Enable Blend" e inserite un valore plausibile. Inserire un valore inferiore nel parametro "Region" porterà le proiezioni verso il centro, mentre un valore più alto le allontanerà verso i bordi. Continuate ad aumentare o diminuire il valore "Region" finché il contenuto nella sovrapposizione sarà lo stesso, quindi abilitate nuovamente il pulsante "Enable Blend".

A questo punto utilizzate lo slider "Blend" per regolare l'intensità della fusione. Un valore più alto renderà la rampa più ripida, mentre un valore più basso creerà una rampa più graduale.

Potreste aver bisogno di apportare delle correzioni alla gamma del contenuto nella zona di sovrapposizione. Questo perché la luce non è percepita linearmente, quindi la luminosità di un pixel impostata a 0.5 potrebbe non essere il doppio di quella impostata per un pixel a 0.25 (se consideriamo i valori della luminosità normalizzati su una scala da 0 a 1). Quasi tutti i dati dei video passano già attraverso delle correzioni della gamma ma, a causa delle differenze tra le tecnologie dei proiettori e data la sovrapposizione di due di essi, potreste trovare necessario apportare piccole correzioni. Questi slider sono utilizzati per compensare o diminuire l'intensità della correzione della gamma.

Infine lo slider “Luminance” controlla la luminosità complessiva della zona di sovrapposizione. Se la zona di fusione è più luminosa o più scura del resto dell’immagine, portare lo slider sotto a 0.5 renderà la zona più scura, mentre portarlo al di sopra la renderà più luminosa.

11 Ottimizzazione

11.1 Introduzione

Quando si lavora in tempo reale risulta incredibile pensare alle sfide che occorre superare: è richiesta un'incredibile precisione, senza considerare tutta la parte relativa ad hardware e software; lavorando su un progetto a 30 FPS, ogni singolo evento da processare, creare e mostrare, deve svolgersi in una finestra di 33 millisecondi. Non è neanche un decimo di secondo! Questo intervallo è persino più piccolo quando si lavora a frequenze di aggiornamento più alte: un progetto che funziona a 60 FPS ha solo 16 millisecondi per renderizzare ogni frame dall'inizio alla fine.

È importante realizzare quanto sia piccola la finestra di opportunità, così da avere cura di preservare ogni singola frazione di millisecondo. Chiedetevi sempre perché un Operatore impieghi un intero millisecondo per completare il suo compito, impuntatevi e cercate di strappare ogni mezzo millisecondo possibile, sapendo che ogni istante fa la differenza. Tutto ciò richiede delle basilari abilità di analisi e di ottimizzazione del progetto.

TouchDesigner impegna pesantemente la CPU e la GPU, ed essere in gradi di controllare quale delle due sia sotto carichi più pesanti è un'abilità importante. Quando si affrontano Network sempre più grandi, sapere dove il sistema rallenta e come ottimizzare gli Operatori per aggirare questi ostacoli può fare la differenza tra consegnare con successo un progetto e non portarlo a termine.

11.2 Trovare il Collo di Bottiglia

L'intero computer può essere considerato come un corso d'acqua. CPU, GPU, RAM e dischi di archiviazione lavorano insieme per creare il prodotto finale. A volte lavorano indipendentemente, ma nella maggior parte dei casi fanno affidamento uno sull'altro, poiché compiono individualmente compiti molto specifici. In un canale la velocità dell'acqua è determinata dal suo componente più debole: a causa di questa natura interconnessa delle componenti, un singolo tratto del corso d'acqua può bloccare un intero progetto, anche se il resto del canale è perfettamente pulito. Questo intoppo, o anello debole della catena, è chiamato collo di bottiglia.

Un esempio di flusso di lavoro con un collo di bottiglia è un progetto che tenta di renderizzare delle semplici geometrie 3D, applicando come texture dei file video. Questo progetto ipotetico consiste di 4 SOP "Box"; ogni faccia dei SOP "Box" ha come texture un file video HAP Q da 1920x1080 pixel. Il computer utilizzato dispone di 32 GB di RAM, un processore dual 8-core, una scheda grafica di fascia alta della serie nVidia Quadro e un singolo hard disk da 5400-RPM.

Quando avviato, il progetto non riesce semplicemente ad essere eseguito da questo sistema. Indipendentemente da quanta RAM, da quanti processori, e da quanto sia costosa la scheda grafica installata, il progetto non può leggere tutti quei file HAP Q da un singolo hard disk da 5400-RPM. Il computer si bloccherà continuamente a causa del disco che non riesce a girare abbastanza velocemente per leggere ogni singolo filmato contemporaneamente. I file HAP Q sono esigenti sui dispositivi di archiviazione, e non importa quanto sia potente il resto del computer, il progetto non funzionerà. La GPU non riesce infatti a leggere i file dall'hard disk, così come il disco non riesce ad iniziare a processare i pixel. L'hard disk, in questo caso, è diventato il collo di bottiglia nel flusso di questo progetto.

Generalmente ci sono tre aree dove compaiono i colli di bottiglia: la GPU, la CPU e gli hard disk.

La GPU è un canale di per sé, e il passaggio più propenso a diventare un collo di bottiglia è quello di pixel shading. Ogniqualvolta si opera sui pixel, utilizzando quasi ogni TOP, il sistema richiede sempre di più dal pixel shader della GPU. Più è alta la risoluzione e più è alta la richiesta. C'è un rapporto di 1:1 tra la risoluzione dei TOP e il loro impatto sulla scheda grafica. Se la risoluzione di un TOP è ridotta di un fattore due, il suo impatto sarà ridotto proporzionalmente. Un modo rapido per controllare se si verificano colli di bottiglia nel pixel shader è abbassare la risoluzione di tutti i TOP generatori, come i TOP "Render" e i TOP "Constant". Se si osserva un immediato aumento nella velocità e nelle performance, allora è chiaro che è presente un collo di bottiglia dovuto al pixel shading.

Quando la scheda grafica è sotto torchio, alcuni TOP apparentemente a caso inizieranno ad avere tempi di processamento più lunghi del normale. Questo risulta evidente quando nel "Performance Monitor" si osservano i tempi di esecuzione dei vari elementi dell'interfaccia utente di TouchDesigner: se all'improvviso i vari elementi dell'interfaccia impiegano più di un millisecondo ad essere calcolati, significa che il Network ha bisogno di essere ottimizzato per eliminare del lavoro dalla GPU.

La CPU è la seconda area dove si verificano più spesso colli di bottiglia. La maggior parte degli Operatori richiedono la CPU per funzionare, cosicché essa può essere sopraffatta velocemente. I colli di bottiglia dei processori tendono ad essere più facili da identificare, perché gli Operatori che impiegano molto tempo ad essere renderizzati sono visibili nel "Performance Monitor". Con il CHOP "Hog" è inoltre possibile verificare quanto margine abbia la CPU. Questo CHOP fa quello che il suo nome suggerisce, e occupa prestazioni della CPU: il parametro "Delay" indica infatti la quantità di secondi che il CHOP "Hog" aggiungerà al tempo di computazione di ogni frame. Se viene creato un CHOP "Hog" e il progetto inizia a perdere frame, significa che la CPU è il nostro collo di bottiglia.

Tutti i file video usano il processore per decodificare i dati dal loro stato compresso; alcuni codec utilizzano più CPU di altri, e leggere simultaneamente molti file video può richiedere più risorse di quanto uno possa immaginare.

Una CPU satura reagisce in modo simile ad una GPU satura, presentando risultati inconsistenti nel "Performance Monitor". Gli Operatori avranno tempi di esecuzione variabili. Questo è perché le loro operazioni vengono avviate ma, prima che finiscano, alla CPU è richiesto di compiere altri processi. La quantità di tempo che gli Operatori impiegano ad aspettare che la CPU torni alla loro operazione è ciò che fa aumentare il tempo di esecuzione. Diversamente da un sovraccarico sulla GPU, un sovraccarico sulla CPU in genere non influenzerà solo i TOP.

I dischi del sistema sono la terza area dove si verificano colli di bottiglia. Operazioni specifiche possono rivelarsi esigenti sui dischi, e quando si prepara un sistema è facile trascurare l'utilizzo di dischi a stato solido (SSD) di alta qualità. Operazioni come leggere e scrivere filmati possono velocemente saturare un disco, a seconda del codec utilizzato. Questo collo di bottiglia apparirà spesso nel "Performance Monitor" sotto a un TOP "Movie", come una riga che recita:

```
1 Waiting for frame for 90 - E:/test.mov from harddrive
```

Dove il numero indica il fotogramma del filmato, mentre il percorso è quello del file video.

11.3 Utilizzare il Performance Monitor

Il “Performance Monitor”, utile quando si cerca di ottimizzare e fare il debug di un progetto, è uno strumento per analizzare il tempo di esecuzione di un frame.

Esistono tre modi per accedere al “Performance Monitor”:

1. Il tasto “F2” sulla tastiera;
2. La combinazione “Alt+Y” sulla tastiera;
3. Fare click su “Performance Monitor” sotto a “Dialogs” nella barra dei menu in alto nello schermo.

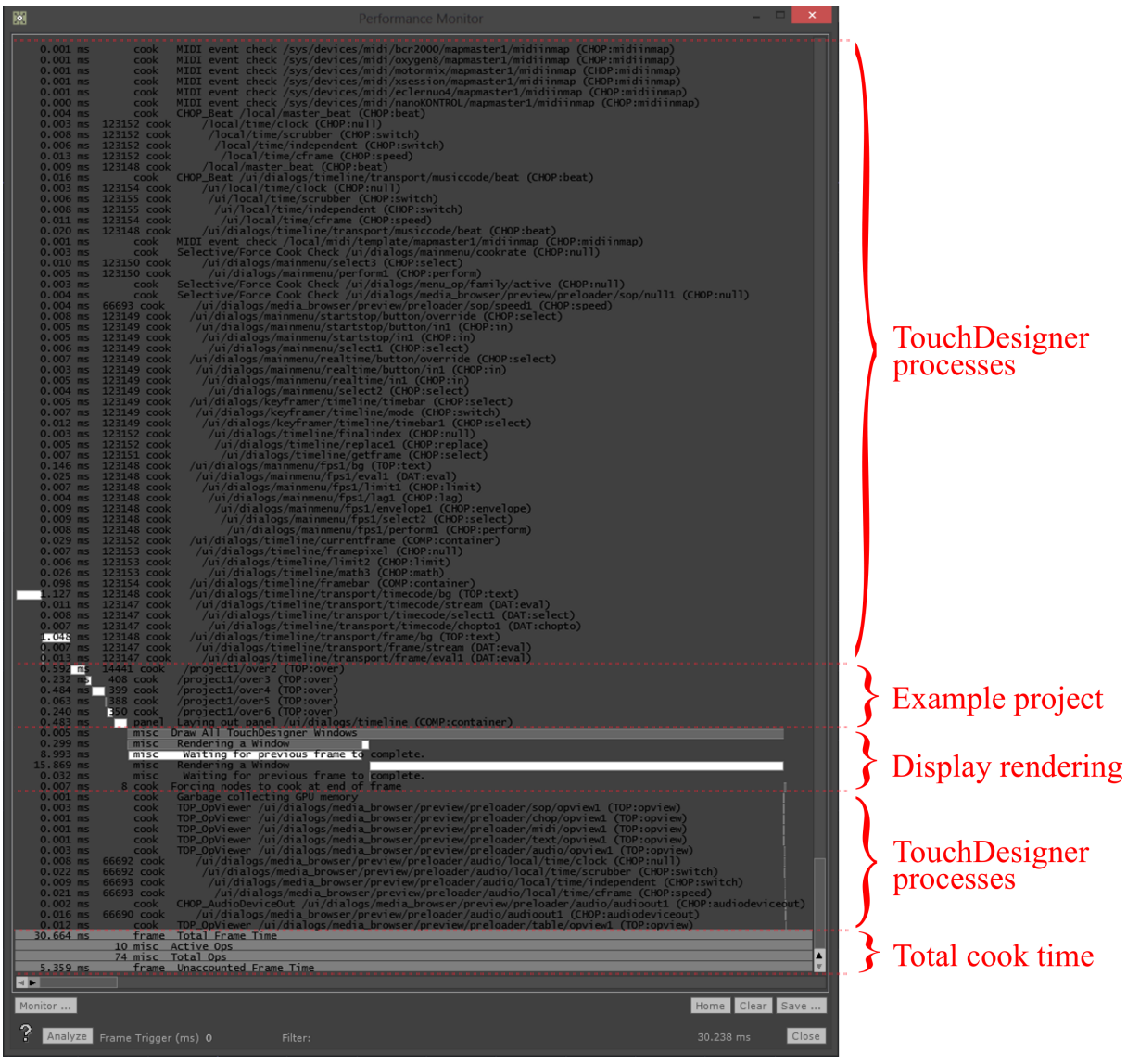
Presenta solo pochi pulsanti, e svolgono funzioni molto semplici, che si spiegano quasi da sole.

Performance Monitor

1. Esegue ed analizza il fotogramma corrente;
2. Cancella i risultati correnti;
3. Salva i risultati correnti in un file di testo;
4. Cambia cosa sta venendo monitorato;
5. Un valore di soglia in millisecondi, che quando superato dal tempo di esecuzione di un frame avvia l'analisi del "Performance Monitor";
6. Filtra i risultati per avere maggior precisione, per esempio solo CHOP, TOP, eccetera.

È importante notare che i tempi di esecuzione sono basati sulla CPU. Questo non significa che nel "Performance Monitor" non sia possibile notare i colli di bottiglia della GPU, ma è giusto sapere che queste letture vengono dalla CPU.

Di seguito è riportata un'analisi di esempio dal "Performance Monitor":



Performance Monitor 2

L'analisi è presa da un semplice progetto con alcuni TOP "Movie In" e alcuni TOP "Over". Ogni frame che viene analizzato conterrà Operatori simili. Questi sono gli Operatori responsabili per il funzionamento e gli elementi dell'interfaccia utente di TouchDesigner.

Osservate le sezioni contrassegnate dello schema precedente per vedere i tempi di esecuzione, e i percorsi, per gli Operatori responsabili delle funzionalità principali di TouchDesigner (interfaccia utente, finestre di dialogo, eccetera), e soprattutto gli Operatori appartenenti al progetto di esempio.

Il tempo di processamento di ogni Operatore è rappresentato da una barra bianca con dimensioni proporzionate al suo contributo al tempo di esecuzione totale di quel frame. Quando i progetti crescono e diventano più ricchi, la sezione ora chiamata "Example project" crescerà per includere tutti gli Operatori del progetto. Questo consente di effettuare l'analisi di un progetto per trovare gli Operatori che occupano troppe risorse del sistema.

In questo esempio i percorsi e i tempi di esecuzione della serie di TOP "Over" possono essere facilmente rintracciati. Sono tutti raccolti nel container "project1", e il loro tempo di esecuzione spazia da 0.06 millisecondi a 0.6 millisecondi.

Una cauta nota a margine: prestate attenzione agli effetti del "Vertical Sync" e a come esso influenzi i tempi di esecuzione percepiti dal "Performance Monitor". La scheda grafica cercherà di sincronizzare gli FPS del progetto e la frequenza di aggiornamento dello schermo: a 30 FPS, anche in un progetto vuoto, il tempo di esecuzione di ogni frame potrebbe essere visualizzato pari a 33 millisecondi; lo stesso effetto può avvenire lavorando a 60 FPS, a parte il fatto che il "Performance Monitor" mostrerà un tempo di esecuzione di 16 millisecondi. Ciò non significa che ogni fotogramma ha effettivamente bisogno di 33 o di 16 millisecondi per renderizzare il frame, ma che il "Vertical Sync" sta cercando di sincronizzare gli FPS di TouchDesigner con la frequenza di aggiornamento dello schermo.

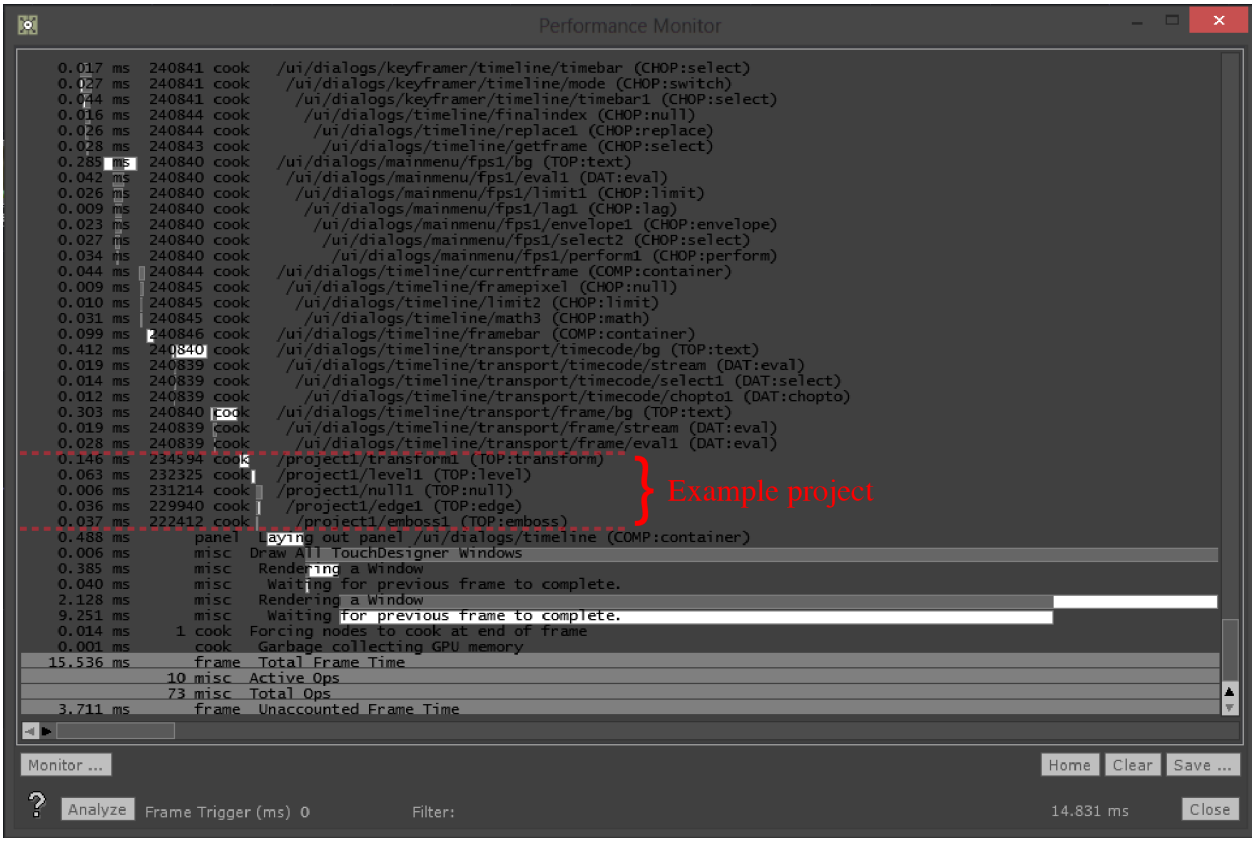
11.4 Esecuzione degli Operatori

È sempre possibile ottenere performance migliori diminuendo il numero di Operatori che vengono eseguiti ad ogni fotogramma. Molti principianti non tengono in considerazione questo fatto quando creano i Network. Va ammesso, tutti si sono trovati a lavorare su prototipi con scadenze ravvicinate, e hanno dovuto creare Network senza poter prendere troppi accorgimenti. Tuttavia, non prendere in considerazione il tempo di esecuzione dei vari nodi può rivelarsi piuttosto rischioso, soprattutto se un progetto non può permettersi di perdere frame.

L'accorgimento principale consiste nel compiere operazioni statiche il prima possibile nel flusso del segnale, per evitare di dover renderizzare tutti gli Operatori ad ogni frame.

Aprite l'esempio "Cooking_1.toe". In questo progetto è presente una semplice grafica che ruota, e successivamente sono utilizzati vari Operatori per conferire all'immagine un aspetto più interessante. Una caratteristica che può aiutare ad ottimizzare un progetto è l'animazione dei fili che connettono gli Operatori: dei fili animati stanno infatti ad indicare che gli Operatori alle due estremità vengono renderizzati ad ogni frame. Partendo dal margine sinistro del Network, il filo tra il TOP "Movie In" e il TOP "Transform" non è animato. Questo perché l'immagine è già stata caricata e rimana statica. Gli Operatori vengono eseguiti solamente quando hanno bisogno di compiere operazioni o cambiamenti, e poiché un'immagine fissa è statica e perciò non cambia, essa non viene eseguita ad ogni frame.

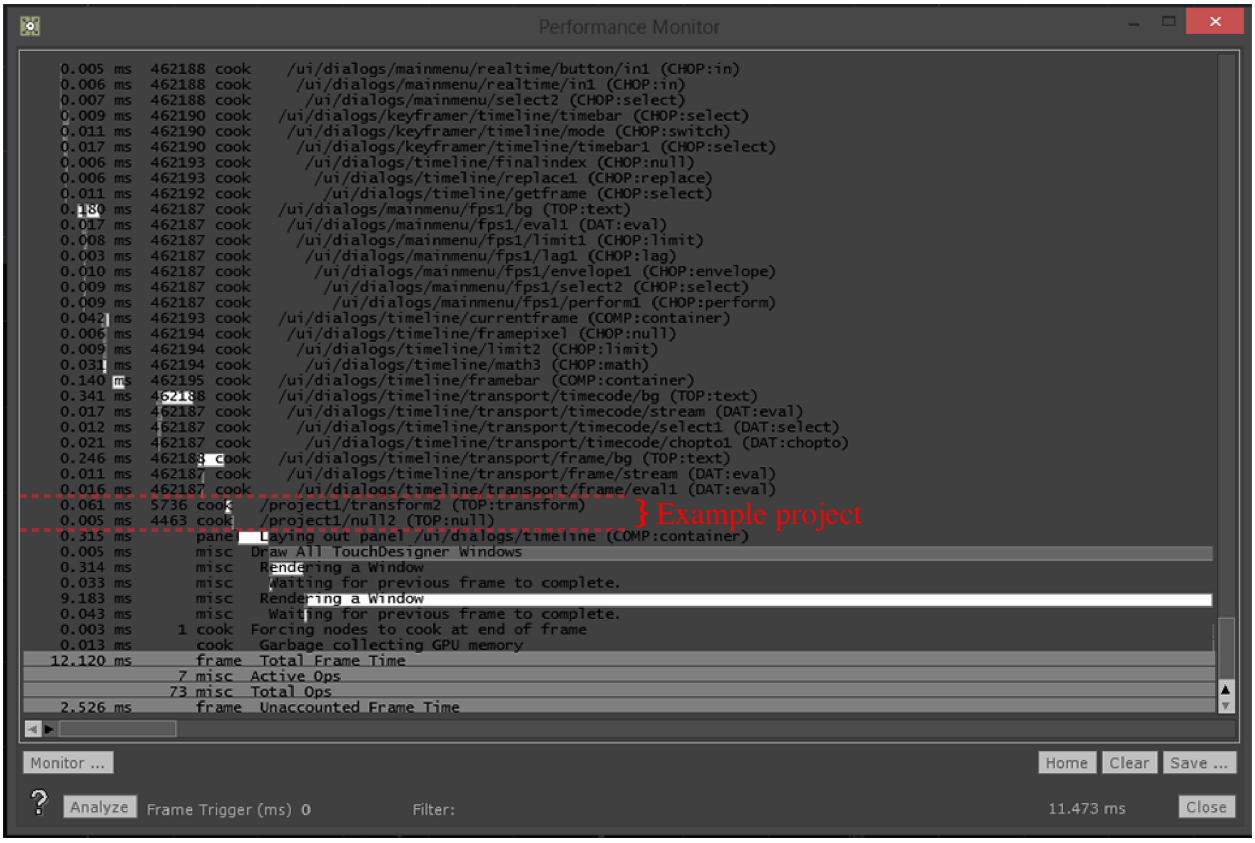
Al contrario, il resto dei fili nel Network sono animati, indicando che tutto dopo il TOP "Movie In" viene calcolato ad ogni fotogramma. Per questo progetto non è un grosso problema, poiché non sta accadendo nulla di estremamente complesso. Entrare nella mentalità di costruire Network efficienti fin da subito può però risparmiare molti mal di testa e molte risorse. Osserviamo questo progetto nel "Performance Monitor".



Operator Cooking

Ignorando tutti gli Operatori necessari al funzionamento di TouchDesigner, rimane un piccolo gruppo di Operatori dedicati a questo esempio. Le operazioni compiute sull'immagine prendono in totale circa 0.25 millisecondi. Come detto in precedenza, le operazioni statiche necessitano di essere eseguite una sola volta, e va notato come molte delle operazioni successive al TOP "Transform" sono naturalmente statiche. Riordiniamole e osserviamo il guadagno in termini di performance.

Aprite l'esempio "Cooking_2.toe". Questo progetto è lo stesso del precedente, a parte il riordinamento degli Operatori. Prima di esaminare il flusso di lavoro più da vicino diamo un occhiata al "Performance Monitor".



Operator Cooking 2

A prima vista sembra che il progetto si sia ristretto! Alcuni degli Operatori che prima erano in elenco sono spariti. Ciò è perché questi Operatori non vengono eseguiti ogni frame. Nell'ultimo esempio il TOP "Transform" stava calcolando una trasformazione ad ogni frame, forzando i TOP dopo di sé ad effettuare le loro operazioni ad ogni frame. In questo esempio tutte le operazioni statiche avvengono all'inizio della catena di segnale, lasciando il TOP "Transform" libero di compiere le sue trasformazioni, senza obbligare altri Operatori a ricalcolare le proprie operazioni o ad essere rieseguito.

Osservando con più attenzione il "Performance Monitor", il solo Operatore che viene eseguito è il TOP "Transform", che impiega 0.061 millisecondi per essere calcolato. Paragoniamo questo all'esempio precedente, dove la serie di operazioni prendeva 0.25 millisecondi per venire portata a termine. È un guadagno incredibile per un cambiamento così piccolo.

Vale la pena notare che messe una di fianco all'altra, le uscite di queste due catene potrebbero non essere esattamente identiche, ma sono così indistinguibili una dall'altra che a molti clienti ed artisti non importerà della differenza, sapendo che l'incredibile risparmio di risorse permetterà di fare tante altre cose.

11.5 Risoluzione

Quando si lavora con texture 2D la risoluzione è estremamente importante, a causa del rapporto quasi 1:1 tra i pixel processati e le risorse utilizzate.

Un modo molto semplice per dimostrare questo fatto è attraverso degli esempi. Aprite il file "Resolution_1.toe". Questo progetto contiene un setup semplice: la farfalla è composta su un quadro più grande, poi usando alcuni LFO vengono modulati l'opacità e il blur prima di comporre il tutto su uno sfondo raffigurante una foresta. Facendo click con il pulsante centrale del mouse su un qualsiasi TOP rivelerà che questo esempio richiede poco più di 100 MB di RAM della GPU. Non è una richiesta pesante per un sistema con 4 o più GB di memoria grafica, ma può aumentare velocemente. Provate a comporre 40 di queste farfalle in tempo reale, e avrete già occupato 4 GB.

Ora paragonate questo all'esempio "Resolution_2.toe". Esso crea esattamente lo stesso risultato, ma utilizzando soli 60 MB di RAM della GPU. È una differenza significativa. Considerate l'esempio precedente di comporre 40 farfalle: usando questo secondo metodo sono necessari solo 2.4 GB circa di RAM della GPU. Tutto questo risparmio per un semplice cambio di risoluzione. L'immagine iniziale della farfalla è solo di 512x512 pixel, e nel primo esempio è immediatamente composta su un quadro da 1920x1080 pixel che è poi modulato. Questo crea uno scenario dove TouchDesigner sta costantemente ridisegnando tutti i 1920x1080 pixel ad ogni frame in cui la farfalla è animata. Anche i pixel "vuoti", che non hanno né colore né canale alfa, vengono costantemente ridisegnati. Nel secondo esempio vengono compiute esattamente le stesse operazioni, ma solo sull'asset iniziale, che ha una risoluzione inferiore. Quest'immagine modulata è poi composta su un quadro da 1920x1080 pixel. Ciò fa risparmiare alla GPU di ridisegnare una grande quantità di pixel, quando solo una piccola porzione richiede di essere processata, risparmiando quindi molta memoria grafica e risorse del sistema.

11.6 Frammentazione della Memoria Grafica

Gli Operatori che usano la GPU spesso occuperanno e manterranno occupate le risorse richieste per i loro compiti finché la loro funzione non verrà completata o modificata.

La frammentazione della GPU è uno dei problemi principali quando si lavora in progetti che hanno molte sorgenti di contenuto differenti. Per esempio quando un TOP "Constant" con una risoluzione di 1280x720 pixel è collegato ad altri dieci Operatori: una volta connesso e calcolato, ogni Operatore metterà da parte la quantità necessaria di memoria grafica richiesta per gestire il calcolo della sua specifica quantità di pixel. Allocata questa memoria, gli Operatori possono operare in modo relativamente efficiente all'interno dello spazio a loro riservato.

Se la risoluzione del TOP "Constant" viene modificata si creerà una reazione a catena, in cui tutti gli altri dieci Operatori dovranno riallocare la quantità corretta di memoria della GPU per il loro funzionamento. Se le risorse degli Operatori vengono riallocate in maniera efficiente, saranno riutilizzati molti degli stessi blocchi di memoria. Se, per un caso sfortunato, non potessero riutilizzare gli stessi bit, dovranno spostarsi in fondo alla memoria. Se questo accade molte volte in rapida successione, la memoria della GPU risulterà frammentata, lasciando il progetto in uno stato di pessime performance mentre la GPU cerca di deframmentare la propria memoria.

I due schemi seguenti cercano di illustrare la frammentazione della memoria nel modo più semplice possibile.

Sono presenti due casi studio, entrambi con parametri simili: abbiamo a disposizione 3 GB di RAM della GPU e occorre caricare tre texture statiche da 1 GB ognuna, mantenendole per un tempo indefinito nella RAM.

Memory Fragmentation

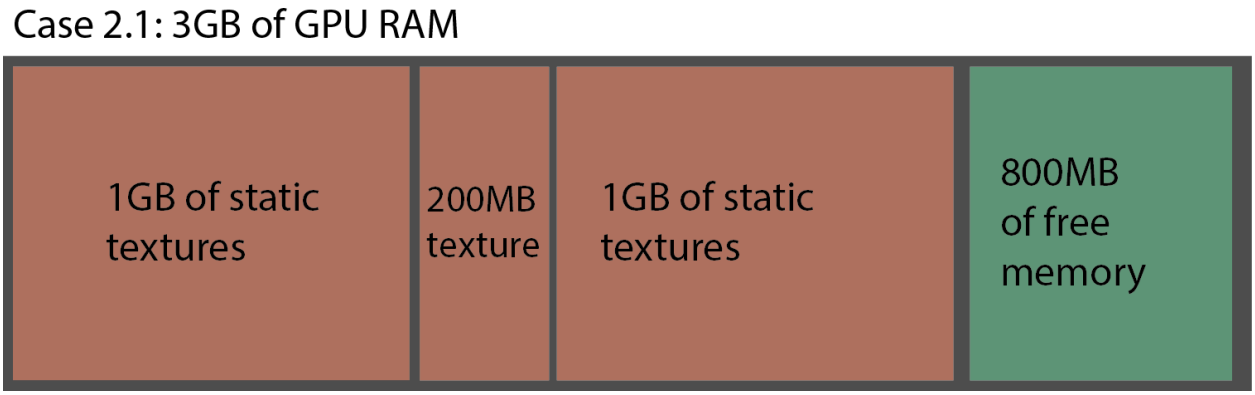
Nel caso 1, i 3 GB di RAM sarebbero in grado di ospitare perfettamente le tre texture statiche da 1 GB. Sono chiamate texture statiche perché, una volta allocate in memoria, non vengono modificate. È l'equivalente di caricare un'immagine in un TOP "Movie In" all'inizio del progetto e lasciarla lì indefinitamente.

Questa è una situazione da mondo ideale, in quanto esistono molti altri processi che utilizzano la RAM della GPU, e ciò comporta che le risorse sono in un continuo flusso, e non ci saranno mai 3 GB di RAM completamente liberi su una scheda grafica che ha solo 3 GB di RAM grafica.

Il caso 2 descrive una situazione dove la memoria è frammentata. Alle tre texture da 1 GB già esistenti è aggiunta una texture da 200 MB. In questo esempio il caricamento e la rimozione dalla memoria avviene nel seguente ordine:

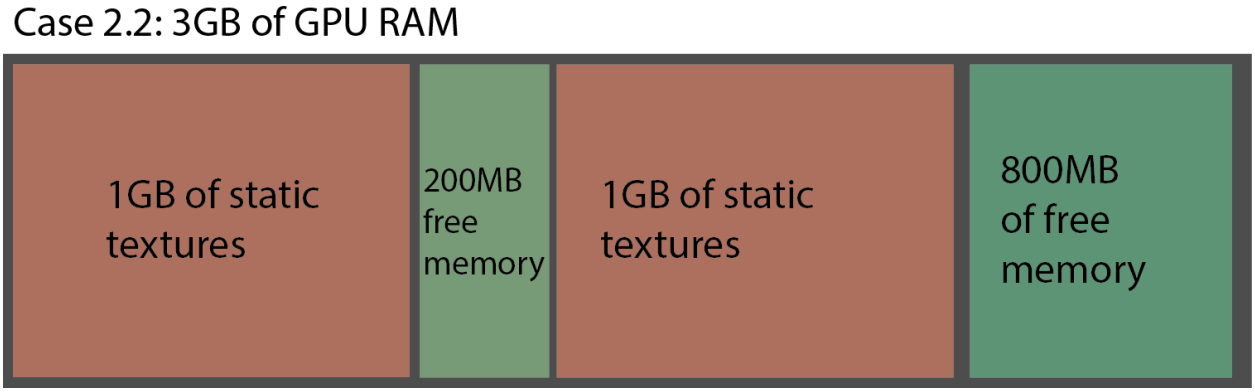
1. Si carica una texture da 1 GB;
2. Si carica la texture da 200MB;
3. Si carica un'altra texture da 1 GB;
4. Si rimuove la texture da 200 MB;
5. Si carica l'ultima texture da 1 GB.

Questo simula una situazione in cui una texture viene caricata, mostrata e quindi rimpiazzata da un'altra texture.



Memory Fragmentation 2

Nello schema "Case 2.1", I passaggi da 1 a 3 sono completati, e rimangono 800 MB liberi. A primo sguardo, questo potrebbe sembrare perfetto, perché se venisse rimossa dalla memoria la texture da 200 MB, ci sarebbe 1 GB di spazio libero per l'ultima texture. Sfortunatamente, non è così che funzionano le schede grafiche.



Memory Fragmentation 3

Come visto in precedenza, nello schema "Case 2.2" il passaggio 4 è stato portato a termine, e la texture da 200 MB è stata rimossa dalla memoria. Ciò che rimane è un primo esempio di frammentazione

della memoria grafica. C'è un totale di 1 GB di memoria libera nella GPU, ma non c'è un singolo blocco da 1 GB per allocare la texture da 1 GB. Le texture da 1 GB già caricate, nella loro condizione statica, non possono essere spostate all'interno della memoria della GPU senza un processo completo di rimozione e ricaricamento in memoria, e poiché la memoria non può essere spostata, i 200 MB di spazio libero sono rimasti intrappolati tra le texture statiche. Questi 200 MB di spazio di allocazione possono essere riempiti con texture da 200 MB o meno, ma non saranno in grado di caricare la terza texture statica da 1 GB.

Il miglior modo per evitare una pesante frammentazione della memoria è provare a ridurre il numero di risoluzioni variabili in un progetto. Quando un asset è scambiato con uno della stessa risoluzione, il più delle volte potrà prendere il suo posto nella memoria.

11.7 Processi di Sistema di Windows

Windows è un sistema operativo complesso, e contiene molti processi e applicazioni legate al sistema che vengono eseguite dietro le quinte. Molti di questi processi e applicazioni possono impattare negativamente sulle performance di un computer, e quindi di TouchDesigner.

Windows è costruito per un ampio mercato di consumatori, che utilizzano principalmente il loro computer durante il giorno. Per questo motivo ci sono molte applicazioni e operazioni del sistema operativo programmate per avviarsi se il computer è lasciato acceso durante la notte. Ciò può rivelarsi problematico per installazioni che devono funzionare 24 ore al giorno. A seconda delle necessità di installazione o di performance, molte differenti applicazioni e operazioni legate al sistema possono, e dovrebbero, essere disabilitate e rimandate.

Applicazioni come programmi di scansione virus e spyware, anche se utili durante l'utilizzo quotidiano, dovrebbero in generale essere evitate quando si utilizza TouchDesigner. Questi programmi possono portare infatti a un numero di problemi, il primo dei quali è la comparsa di pop-up: molti di questi software mostrano pop-up di promemoria e di notifica ad intervalli regolari. In situazioni sfortunate, questi possono sovrapporsi con il contenuto mostrato e coprire le uscite durante le performance e le installazioni. Queste applicazioni possono influire negativamente anche sui dischi, poiché spesso avviano scansioni del sistema alla ricerca di virus e malware, occupano cicli di lettura e scrittura degli hard drive. Queste due problematiche si aggiungono alla richiesta sulla CPU esercitata da molte di queste applicazioni sempre in funzione.

12 GLSL

12.1 Introduzione

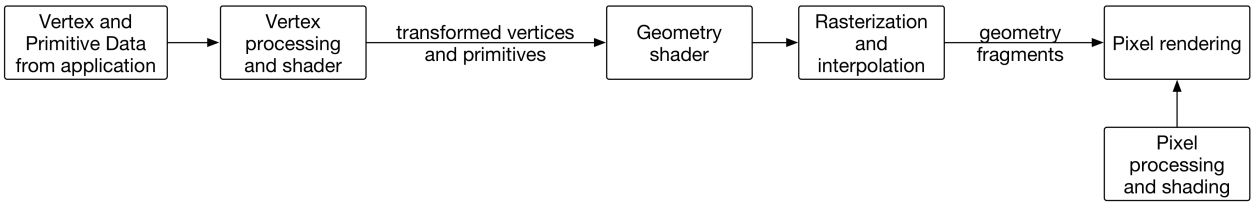
GLSL, o OpenGL Shading Language, è uno strumento incredibile da conoscere: con un po' di codice permette di programmare direttamente sulla scheda grafica.

Molti evitano di imparare GLSL perché credono sia utile solo per lavorare con grandi texture (con risoluzioni di 4k e oltre), per gestire complesse scene generative in 2D, o per creare e manipolare geometrie 3D, ma in realtà può essere sfruttato in moltissimi aspetti della programmazione di ogni giorno, sia per ottimizzare e creare workflow di compositing incredibilmente efficienti, anche per contenuti HD standard, oppure per creare semplici sfondi generativi per esperienze interattive.

Lo scopo di questo capitolo è introdurre il lettore ad alcuni semplici workflow e tecniche GLSL, senza presumere alcuna esperienza precedente con tale linguaggio di programmazione.

12.2 Tipologie di Shader e Flussi di Rendering

Le due principali tipologie di shader che possono essere programmate in TouchDesigner sono lo shader Vertex e lo shader Pixel (o Fragment). In un flusso di rendering standard, essi sono eseguiti nell'ordine mostrato nello schema di seguito:



GLSL Pipeline

Esiste anche lo shader Geometry, che può essere posizionato tra gli shader Vertex e Pixel per rendering su più livelli e trasformazioni con feedback, ma non è utilizzato comunemente.

Nel flusso di lavoro riportato in precedenza, l'applicazione passa i vertici e i dati delle primitive alla GPU. Questo include informazioni come la posizione dei vertici, le coordinate delle texture, i colori, le normali, eccetera. Questa informazione può essere processata programmaticamente e lavorata utilizzando uno shader Vertex.

Quando lo shader Vertex ha finito di processare i dati dei vertici e delle primitive, la geometria viene rasterizzata. Durante questo processo la geometria passa attraverso vari processi, come il clipping, il culling, le trasformazioni nello spazio della finestra, eccetera. Questi passaggi preparano i frammenti che saranno processati dallo shader Pixel.

Lo shader Pixel prende questi frammenti, li processa e restituisce colore e valori di profondità per ogni pixel.

Per uno sguardo più approfondito al flusso di rendering, è possibile trovare una spiegazione più rigorosa sul sito OpenGL:

http://www.opengl.org/wiki/Rendering_Pipeline_Overview[10]

[10] http://www.opengl.org/wiki/Rendering_Pipeline_Overview

12.3 Debugging

Prima di immergerci troppo a fondo nel mondo GLSL è importante sapere come risolvere problemi di compilazione. Ci sono due segnali che indicano un errore di compilazione.

Il primo è un pattern a scacchiera blu e rosso: quando si lavora con il MAT "GLSL" questo pattern coprirà la geometria stessa; quando invece si usa GLSL con dei TOP, questo pattern a scacchiera riempirà il viewer dell'Operatore e sarà in uscita da quel TOP.

Il secondo segnale di un errore di compilazione è un'icona gialla a forma di triangolo presente sull'Operatore. Se si fa click e si tiene premuto il tasto sinistro del mouse sul segnalino, l'errore reciterà:

```
Warning: The GLSL Shader has compile error
```

Per diagnosticare questi problemi, è richiesto un DAT "Info". Create un DAT "Info" e impostate il parametro "Operator" per fare riferimento all'Operatore GLSL con l'errore. Nel DAT "Info", se non ci sono errori, saranno mostrate solo poche linee, e una di quelle riporterà "Compiled Successfully" o "Linked Successfully". Se invece ci sono errori, comparirà qualcosa di simile a:

```
Pixel Shader Compile Results:
0(3) : error C0000: syntax error, unexpected '=', expecting '::' at token '='
```

Per capire velocemente cosa indichino questi errori, la prima cosa è individuare il numero della riga in cui si verificano. In questo caso "o(3)" indica che il compilatore incontra degli errori alla linea 3. In molte situazioni questo può significare che l'errore effettivo avviene una o due righe prima, ma il compilatore non si blocca fino alla prima linea in cui le cose iniziano a non avere più senso per lui.

La sezione successiva evidenzia che è presente un errore di sintassi, e quindi sono riportate altre informazioni relative all'errore.

Per i principianti, la maggior parte di questi errori accadrà a causa di punti e virgola dimenticati alla fine delle linee di codice, o per errori in cui si tenta di mettere un tipo di dati all'interno di un'altra tipologia incompatibile.

12.4 Shader in 3D

Iniziamo osservando alcune semplici scene 3D che utilizzano gli shader Vertex e Pixel.

Aprite l'esempio "Basic_3D.toe".

È un semplice setup di rendering dove lo shader Vertex non fa altro che trasmettere i vertici allo shader Pixel, che quindi colora tutto di bianco. Osserviamo più approfonditamente ogni shader, iniziando dallo shader Vertex:

```
1 void main(){
2         gl_Position = TDWorldToProj(TDDeform(P));
3 }
```

Poiché questo capitolo è scritto per chi non ha alcuna esperienza con GLSL, evidenziamo alcune note riguardo la sintassi basilare in C e le regole che devono essere seguite, poiché GLSL è molto simile a C.

La prima cosa è che i punti e virgola devono essere aggiunti alla fine di ogni linea di codice.

La seconda cosa è che la maggior parte del lavoro di coding avverrà all'interno del loop main(). Per i principianti l'unica cosa al di fuori del loop main() sarà la dichiarazione delle variabili quali i segnali e gli attributi in ingresso e in uscita.

Quando si crea uno shader vuoto, è una buona abitudine iniziare inserendo il loop main() come segue:

```
1 void main(){
2
3 }
```

In questo shader Vertex è presente una sola linea di codice:

```
1 gl_Position = TDWorldToProj(TDDeform(P));
```

Analizziamolo nel dettaglio, procedendo da destra verso sinistra, poiché avvengono molte cose al suo interno.

"P" in questo caso indica la posizione dei vertici. In TouchDesigner ci sono alcuni attributi che vengono dichiarati di default: per esempio "P" per la posizione dei vertici, "N" per le normali, "Cd" per i colori, e "uv" per i livelli di coordinate delle texture (sono degli array accessibili con degli indici, per esempio uv[0], uv[1], eccetera). Questi valori possono essere richiamati facilmente, come abbiamo fatto per "P".

Nella linea di codice, "P" è l'input di una funzione chiamata "TDDeform()". Questa funzione prende la posizione dei vertici dallo spazio dell'oggetto e la porta nello spazio globale.

Pensate allo spazio in 3D dell'oggetto come se ogni elemento avesse il suo sistema di coordinate individuale, con origine in (0,0,0). Dopo aver processato questi vertici, occorre trasformarli e integrarli nello spazio globale che, come implica il suo nome, è lo spazio completo del mondo considerato. Il mondo, in questo caso, è il sistema di coordinate comuni, con un solo punto di origine (0,0,0), dove tutti gli oggetti esistono inseme.

Una volta che le posizioni dei vertici sono riportate nello spazio globale, esse vengono trasmesse alla funzione "TDWorldToProj()". Questa funzione prende le coordinate dello spazio globale e le riporta nello spazio proiettato, che prende tutte le coordinate dallo spazio globale e le restituisce dal punto di vista di una telecamera.

Questi valori sono quindi assegnati alla variabile "gl_Position", che è un'uscita preassegnata per i vertici trasformati.

Come mostrato nello schema del flusso di rendering, questi vertici trasformati sono processati e trasmessi allo shader Pixel, dove vengono loro assegnati dei valori per i colori.

In questo esempio lo shader Pixel imposta su bianco il colore in uscita di tutti i frammenti. Il codice è il seguente:

```
1 out vec4 fragColor;
2 void main(){
3         fragColor = vec4(1,1,1,1);
4 }
```

In modo simile allo shader Vertex, tutto il codice a parte la dichiarazione delle uscite si trova all'interno del loop main(), e i punti e virgola sono presenti al termine di ogni riga di codice.

Partendo dall'alto, la prima riga di codice recita:

```
1 out vec4 fragColor;
```

Questa linea imposta l'uscita come una variabile chiamata "fragColor". Un "vec4" è un vettore con 4 componenti, che in questo scenario corrispondono all'uscita rossa, verde, blu e al canale alfa. In GLSL, ogniqualvolta si dichiara un ingresso o un'uscita occorre dichiararne anche il tipo, per esempio specificando che sarà un "vec4". In modo simile, gli ingressi saranno preceduti da un "in" e le uscita da un "out".

La riga di codice all'interno del loop main() è la seguente:

```
1 fragColor = vec4(1,1,1,1);
```

Poiché "fragColor" è già stata dichiarata come uscita, questa riga scrive i valori dei pixel direttamente sull'uscita. La sezione "vec4(1,1,1,1)" crea un vettore a 4 componenti con valori (1,1,1,1) e lo assegna a "fragColor". L'espressione "vec4" deve precedere la lista di valori, perché "fragColor" è già stato dichiarato come un "vec4", quindi è possibile assegnargli solamente un valore che sia di tipo "vec4".

Poiché i 4 componenti dell'uscita corrispondono ai canali RGBA, il vettore (1,1,1,1) imposterà tutti i canali a 1, rendendo in pratica l'uscita bianca per ogni pixel.

Ora, con una comprensione basilare del funzionamento degli shader in una scena 3D, aggiungiamo alcuni livelli di complessità. Per prima cosa ridimensioniamo il cubo. È possibile fare ciò semplicemente moltiplicando ognuna delle posizioni dei vertici, o "P", per un valore.

Aprite l'esempio "Basic_3D_scaled.toe".

Solo poche linee dello shader Vertex sono diverse:

```
vec3 scaledP;

void main(){
        scaledP = P * 0.5;
        gl_Position = TDWorldToProj(TDDeform(scaledP));
}
```

La prima aggiunta è la dichiarazione di una variabile chiamata "scaledP" che verrà usata successivamente. Essa sarà un vettore a 3 componenti, poiché lavoreremo con la posizione dei vertici ("P"), che è a sua volta un vettore a 3 componenti.

Dopo aver definito questa variabile, essa può essere utilizzata all'interno del codice. La successiva riga che è stata aggiunta riporta:

```
scaledP = P * 0.5;
```

Questa linea prende le posizioni dei vertici e moltiplica tutti i valori per 0.5, rendendo in pratica il cubo grande la metà. In modo analogo, un valore di 2 renderebbe il cubo due volte più grande delle sue dimensioni originali.

Questo nuovo valore, "scaledP", prende quindi il posto di "P":

```
gl_Position = TDWorldToProj(TDDeform(scaledP));
```

In modo simile, per trasformare il cubo invece di scalarlo, è necessario aggiungere o sottrarre valori seguendo i vari assi.

Aprite l'esempio "Basic_3D_transform.toe".

In questo progetto il codice nello shader Vertex è stato modificato come segue:

```
vec3 transformedP;

void main(){
        transformedP = P + vec3(1,0,0);
        gl_Position = TDWorldToProj(TDDeform(transformedP));
}
```

Ciò è molto simile all'esempio precedente: inizia dichiarando un vettore a 3 componenti chiamato "transformedP" ma, nella prima riga del loop main(), invece di moltiplicare le posizioni dei vertici per 0.5, il nuovo vettore a 3 componenti viene sommato ad esse.

La ragione di usare un "vec3" in questo esempio è che se fosse semplicemente stato aggiunto 0.5 a "P", tale valore sarebbe stato aggiunto all'asse x, all'asse y e all'asse z. Creando un "vec3" possiamo invece controllare ogni specifico asse. In questo esempio aggiungere un vettore "vec3(1,0,0)" aggiungerà 1 solamente all'asse x, lasciando gli assi y e z inalterati. In pratica, sposta il cubo di una unità verso la destra della telecamera.

Potremmo facilmente cambiare da addizione a sottrazione per spostare il cubo nella direzione opposta.

Facciamo ora riferimento dallo shader Vertex ad un CHOP "LFO" per animare il movimento.

Aprite l'esempio "Basic_3D_LFO.toe".

Notiamo come sia stato aggiunto un CHOP "LFO". Il valore del suo canale è stato quindi esportato al parametro "value0x" del MAT "GLSL", nella schermata "Vectors 1" della finestra dei parametri. Gli è stato quindi dato un nome, in questo caso "lfoCHOP". Ciò significa che il valore può ora essere accessibile dall'interno degli shader Vertex e Pixel.

Il codice nello shader Vertex è stato modificato come segue:

```
vec3 transformedP;
uniform float lfoCHOP;

void main(){
        transformedP = P + vec3(lfoCHOP,0,0);
        gl_Position = TDWorldToProj(TDDeform(transformedP));
}
```

La prima aggiunta è situata nella seconda riga, dove è stata dichiarata una variabile "uniform". Una "uniform" è una variabile globale di GLSL utilizzata principalmente per parametri che un utente o un programma trasmetteranno allo shader.

In questo esempio il canale del CHOP "LFO" è un valore decimale, perciò è stato aggiunto "float" dopo "uniform". Il nome della "uniform" è importante, perché deve corrispondere al nome inserito nel parametro "Uniform Name" del MAT "GLSL". Nel MAT "GLSL" abbiamo chiamato la "uniform" "lfoCHOP", quindi per accedervi deve essere utilizzato lo stesso nome.

L'unico altro cambiamento è che dove in precedenza veniva aggiunto 1 all'asse x delle posizioni dei vertici, ora è presente il valore di "lfoCHOP".

```
transformedP = P + vec3(lfoCHOP,0,0);
```

Con questi piccoli cambiamenti, un CHOP controlla la posizione sull'asse x dello shader. Gli shader Pixel funzionano in modo molto simile agli shader Vertex. Assegniamo il CHOP "LFO" dell'esempio precedente come controllo del canale rosso dei colori in uscita.

Aprite l'esempio "Basic_3D_red.toe".

In questo progetto il CHOP "LFO" sta controllando il canale rosso dello shader Pixel nello stesso modo in cui precedentemente controllava la trasformazione sull'asse x.

Il codice nello shader Pixel è il seguente:

```
out vec4 fragColor;
uniform float lfoCHOP;

void main(){
        fragColor = vec4(lfoCHOP,0,0,1);
}
```

In modo simile all'esempio relativo alla trasformazione sull'asse x, gli unici passaggi necessari sono stati dichiarare la "uniform" e quindi assegnarla ad un parametro, in questo caso al primo componente del vettore. Per portare questo esempio un passo più in là, prendiamo un TOP "Ramp" come texture, invece di utilizzare un singolo colore.

Aprite l'esempio "Basic_3D_texture.toe".

La prima cosa fatta è stato creare un TOP "Ramp" e assegnarlo al MAT "GLSL". Il TOP deve essere assegnato nella schermata "Samplers 1" nella finestra dei parametri. Allo stesso modo in cui il CHOP "LFO" necessitava di uno "Uniform Name", il TOP "Ramp" ha bisogno di un "Sampler Name", che in questo caso è "rampTOP".

Quindi è stato aggiunto un SOP "Texture" dopo il SOP "Box" per assegnare le coordinate della texture.

Nello shader Vertex sono state aggiunte alcune linee di codice per controllare le coordinate. Le linee aggiunte sono le seguenti:

```
vec3 transformedP;
uniform float lfoCHOP;
out vec2 texCoord0;

void main(){
        transformedP = P + vec3(lfoCHOP,0,0);
        gl_Position = TDWorldToProj(TDDeform(transformedP));

        vec3 texCoord = TDInstanceTexCoord(uv[0]);
        texCoord0.st = texCoord.st;
}
```

La prima linea aggiunta è:

```
out vec2 texCoord0;
```

Questa riga crea un vettore a 2 componenti chiamato “texCoordo”, che può essere utilizzato all0interno dello shader Pixel. In questo caso saranno le coordinate UV della texture.

La seconda riga aggiuntiva recita:

```
texCoord0.st = uv[0].st;
```

Essa prende il vettore “texCoordo” dichiarato in precedenza e gli assegna la variabile “uv”, preesistente in TouchDesigner, che come detto in precedenza è dichiarata di default e contiene le coordinate UV della texture (in modo simile alla posizione dei vertici “P”).

Il termine “.st” assegna i due valori contenuti nel vettore con componenti “uv” alle due componenti di “texCoordo”. Come menzionato prima, se “.xyzw” sono utilizzati per le posizioni dei vertici, “.stpq” sono spesso usati per le coordinate delle texture. Sono utilizzati per convenzione, cosicché le stesse lettere (come XYZW e RGBA) non indichino più cose diverse. Potreste anche incontrare “.uv” al posto di “.st”, a seconda del software utilizzato.

Con queste due linee di codice extra, il SOP “Box” è ora texturizzato dal TOP “Ramp”.

12.5 Shader in 2D

Anche se non si ha intenzione di spendere molto tempo ad imparare il linguaggio GLSL per il 3D, lo shader Pixel può tornare utile come strumento per realizzare visual generativi o compositare texture in 2D. Semplici shader in GLSL possono essere particolarmente comodi quando si compositano texture, poiché sono in grado di far risparmiare molta memoria grafica quando si compiono lavori ripetitivi.

Osserviamo un esempio in cui sommiamo insieme due TOP, in modo simile al TOP "Add".

Aprite il progetto "Basic_2D_add.toe".

Usando il TOP "GLSL" molti TOP possono essere compositati in ingresso, campionati ed effettati utilizzando lo shader Pixel in GLSL. In questo esempio un TOP "Movie In" e un TOP "Constant" sono inseriti in un TOP "GLSL". Il codice GLSL è nello shader Pixel, che li somma:

```
out vec4 fragColor;

void main(){
        vec4 in1 = texture(sTD2DInputs[0], vUV.st);
        vec4 in2 = texture(sTD2DInputs[1], vUV.st);
        fragColor = in1 + in2;
}
```

Lavorare con gli shader Pixel nella famiglia dei TOP è molto simile a lavorare con loro utilizzando un MAT "GLSL". È ancora presente un loop main(), la dichiarazione avviene subito prima di esso, e occorre inserire punti e virgola alla fine di ogni riga di codice.

La prima differenza da notare è come sono gestiti gli ingressi. In questo esempio sono presenti due input, gestiti da due righe:

```
vec4 in1 = texture(sTD2DInputs[0], vUV.st);
vec4 in2 = texture(sTD2DInputs[1], vUV.st);
```

Per avere accesso ad un TOP in entrata occorre dichiarare una variabile a quattro componenti per i canali RGBA. La funzione "Texture()" è utilizzata per campionare la texture: essa necessita di due variabili, la prima con la texture che deve campionare, in questo caso i TOP in ingresso, la seconda contenente le coordinate della texture da campionare.

In questo caso, poiché vanno campionati i TOP in ingresso, è stata usata la matrice già presente chiamata "sTD2DInputs". Questa matrice viene referenziata con degli indici: nell'esempio ci sono due ingressi, perciò per accedere al primo input si usa "sTD2DInputs[0]", mentre per accedere al secondo si utilizza "sTD2DInputs[1]".

Per accedere alle giuste coordinate delle texture si utilizza “vUV.st”. “vUV” è una variabile predichiarata che contiene le coordinate delle texture dei pixel. Come già detto, “.st” è utilizzato per richiamare le prime due componenti del vettore.

Una volta che da ciascun ingresso si è fatto riferimento al pixel appropriato, si procede a sommare insieme i pixel, e quindi trasmetterli a “fragColor”, che è l’uscita definita come:

```
fragColor = in1 + in2;
```

In modo così semplice è stata replicata la funzionalità del TOP “Add”. Può sembrare un metodo complicato di sommare due texture, ma i benefici di qualcosa di così semplice iniziano a rendersi evidenti quando si utilizzano flussi di lavoro più complicati, o il TOP “GLSL Multi”, che è fondamentalmente uguale al TOP “GLSL”, ma con un numero infinito di ingressi (limitato solo dalla scheda grafica).

Aprite l’esempio “Basic_2D_multi.toe”.

Questo progetto espande quello precedente sommando insieme più sorgenti. Ciò è fatto connettendo tutte le fonti in un TOP “GLSL Multi”, e quindi accedendo ad esse nello stesso modo in cui prima si accedeva ai due ingressi.

Nello shader di questo esempio ogni ingresso prende l’indice successivo di “sTD2DInputs” e lo assegna ad un altro “vec4”, e dopodiché sono tutti sommati insieme.

Ancora una volta, questo può essere utile, ma anche un TOP “Composite” può sommare ingressi multipli, quindi spingiamoci un passo più in là.

Aprite l’esempio “Basic_2D_composite.toe”.

Questo progetto spinge oltre l’esempio precedente. Con tutti gli ingressi definiti, invece di limitarsi a sommare tutti gli input, il codice GLSL compie un misto di addizioni, sottrazioni e moltiplicazioni. Ricordare l’ordine delle operazioni quando si lavora in questo modo è fondamentale.

Tale procedimento è incredibilmente utile quando si lavora con un grande numero di texture, anche se solo da 1920x1080 pixel, poiché compiere un processo simile con i TOP richiederebbe molti TOP “Composite”, o un gran numero di TOP “Multiply”, “Add” e “Subtract”. Essere in grado di manipolare tutte queste texture, anche in modi così semplici, in una volta sola, può far risparmiare parecchia memoria grafica.

Trasformare le texture è molto simile a trasformare i vertici in uno shader Vertex. Osserviamo un esempio.

Aprite il file “Basic_2D_transform.toe”.

Questo progetto prende quello precedente e sposta alcune delle texture. Ciò è fatto sommando, sottraendo, moltiplicando e dividendo le coordinate delle texture. Esaminiamo di seguito il codice dell’esempio:

```
out vec4 fragColor;

void main(){
        vec4 in1 = texture(sTD2DInputs[0], vUV.st * 0.5);
        vec4 in2 = texture(sTD2DInputs[1], vUV.st);
        vec4 in3 = texture(sTD2DInputs[2], vUV.st - vec2(0.5,0.25));
        vec4 in4 = texture(sTD2DInputs[3], vUV.st);
        vec4 in5 = texture(sTD2DInputs[4], vUV.st + vec2(0.5,0.));

        fragColor = in1 + in2 * in3 - in4 * in5;
}
```

In questo caso tre delle texture subiscono trasformazioni. La prima viene scalata di un fattore due. Per fare ciò "vUV.st" è moltiplicato per 0.5. Questo può sembrare errato, ma ricordate che "vUV" rappresenta le coordinate delle texture, perciò quando le coordinate degli assi x e y sono spostate a destra, l'immagine si sposta a sinistra. Immaginate di avere un dito che punta al centro di un foglio di carta. Se la posizione del dito è spostata a destra, l'immagine sarà più a sinistra del dito rispetto a prima. Questa è un'enorme semplificazione, ma dovrebbe aiutare a far passare il concetto.

Il terzo input, "in3", è stato traslato sia sull'asse x che sull'asse y sottraendo 0.5 dall'asse x e 0.25 dall'asse y.

L'ultimo ingresso, "in5", è stato traslato sull'asse x aggiungendo 0.5 in quella direzione.

Entrambe queste trasformazioni seguono lo stesso ragionamento fatto per la prima. Per esempio, quando si aggiunge 0.5 alle coordinate delle texture sull'asse x, viene sommato alle coordinate delle texture, e non alle posizone dell'immagine. Perciò quando le coordinate delle texture si muovono di 0.5 sull'asse x, l'immagine appare spostata a sinistra.

L'ultimo esempio consisterà nel prendere le uscite del caso precedente e trasmetterle a differenti buffer, invece di comporle insieme.

Aprite il progetto "Basic_2D_buffers.toe".

Lo shader è il seguente:

```
out vec4 fragColor[5];

void main(){
        vec4 in1 = texture(sTD2DInputs[0], vUV.st * 0.5);
        vec4 in2 = texture(sTD2DInputs[1], vUV.st);
        vec4 in3 = texture(sTD2DInputs[2], vUV.st - vec2(0.5,0.25));
        vec4 in4 = texture(sTD2DInputs[3], vUV.st);
        vec4 in5 = texture(sTD2DInputs[4], vUV.st + vec2(0.5,0.));

        fragColor[0] = in1;
```

```
11          fragColor[1] = in2;
12          fragColor[2] = in3;
13          fragColor[3] = in4;
14          fragColor[4] = in5;
15
16  }
```

La prima riga dello shader è stata modificata così che l'uscita "fragColor" sia ora una matrice. Ogni componente della matrice può ospitare una texture differente, come fatto alla fine dello shader. A questo punto, con il parametro "# of Color Buffers" del TOP "GLSL Multi" impostato a 5, un TOP "Render Select" può essere utilizzato per selezionare individualmente i singoli buffer dall'interno del TOP "GLSL Multi". Per fare ciò, il parametro "Render or GLSL TOP" deve fare riferimento al TOP "GLSL Multi", e il parametro "Color Buffer Index" deve corrispondere al buffer voluto.

Assegnare le texture a dei buffer è simile ad assegnarle ad una singola uscita, con la semplice aggiunta dell'indice del buffer:

```
1  fragColor[0] = in1;
```

Il numero nelle parentesi quadre fa riferimento al buffer in cui si scrive. Nell'esempio precedente si sta scrivendo nel primo buffer.

12.6 Importare gli Shader da Shadertoy

Capitolo di Matthew Hedlin

Questa sezione illustrerà le basi del trasferimento di uno shader da Shadertoy.com a TouchDesigner. Raccomandiamo di utilizzare un editor esterno, come Sublime Text, Atom o Notepad++, poiché sarà essenziale l'utilizzo di funzioni come trova e sostituisci.

Shadertoy API

Quando si importa uno shader da Shadertoy in TouchDesigner, possiamo utilizzare il nostro buon senso per trovare o per realizzare sorgenti simili a quelle in ingresso a Shadertoy, oppure possiamo scaricare direttamente i file utilizzati sul sito sfruttando l'apposita API. Per scaricare i vari elementi, occorre registrare un account Shadertoy e creare una "App Key".

Una volta eseguito il login in Shadertoy, facciamo click su "Profile" in alto a destra, quindi su "your Apps" nella sezione "Config". Scegliamo un nome e una descrizione e confermiamo facendo click sul tasto "Create".

La "App Key" comparirà nella sezione "Manage our Apps". Ora, copiamo nel nostro browser l'indirizzo "https://www.shadertoy.com/api/v1/shaders/MdlGDM?key=" e alla fine inseriamo la "App Key".

Dopo aver premuto invio, otterremo un oggetto JSON con una chiave chiamata "inputs". In questo esempio, l'indirizzo riportato necessita del file chiamato "tex09.jpg".

Inserendo quindi "https://www.shadertoy.com/presets/tex09.jpg" come URL nel browser, sarà possibile vedere e scaricare la texture richiesta.

Swizzling

Lo "swizzling" è la tecnica utilizzata per accedere alle componenti di un vettore, che incontreremo diverse volte nel corso di questi esempi.

Se per esempio fosse presente una variabile "vec4 color", i 4 valori di "color" sarebbero rappresentati come "color.rgba". Se volessimo accedere solo ai primi due valori del vettore potremmo usare "color.rg", se invece volessimo riordinare i valori del vettore basterebbe scrivere "color.bgar".

Esistono scritture equivalenti a ".rgba" che fanno esattamente la stesa cosa, ma vengono generalmente utilizzate in altri contesti, per fornire una maggior chiarezza nella lettura del codice.

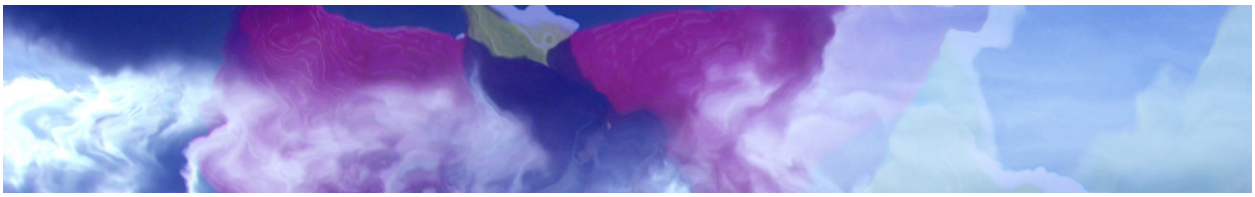
In generale, quando ci si riferisce alle coordinate di una texture si utilizza ".stpq", mentre quando si fa riferimento a delle coordinate 3D si usa ".xyzw".

Queste convenzioni restituiscono lo stesso risultato, ma non possono essere combinate: per fare riferimento ai valori di un "vec4 a" possiamo usare "a.xyzw", oppure "a.stpq", ma non possiamo scrivere "a.stzw", poiché restituirebbe un errore.

Risoluzione dei problemi

Un errore incontrato comunemente quando si convertono shader da Shadertoy è collegato all'estensione di default delle UV. Se vi accorgete che il risultato del vostro shader si discosta da quelli visti sul sito, provate a impostare il parametro "Input Extend Mode UV" del TOP "GLSL" su "Repeat".

Esempio 1: Waterly Video - Test



Example 1: Waterly Video - Test

Shader scritto da: FabriceNeyret2[11]

https://www.shadertoy.com/view/MdlGDM

Setup

Iniziamo creando un TOP "GLSL" e un DAT "Info", quindi inseriamo il nome del TOP "GLSL" nel parametro "Operator Field" del DAT "Info".

Nella sezione "Common" del TOP "GLSL" cambiamo la "Output Resolution" in "Custom" e inseriamo "1280" e "720" nei campi "Resolution".

Copiamo il codice da Shadertoy e incolliamolo nel DAT "glsl1_pixel", sostituendo il codice che era presente di default.

Ora abbiamo bisogno di impostare le sorgenti; per questo esempio ci limiteremo a creare due "TOP "Movie File In" e a selezionare due immagini con la stessa risoluzione del TOP "GLSL" (1280x720 pixel): "Mettler.3.jpg" e "Trillium.jpg".

Funzione Principale e suoi Parametri

All'interno di Shadertoy la funzione principale con i suoi parametri è:

```
mainImage( out vec4 fragColor, in vec2 fragCoord )
```

Ma noi la modificheremo in:

```
main()
```

Per sostituire l'elemento "fragColor" che abbiamo rimosso, dobbiamo inserire all'inizio del codice:

```
layout(location = 0) out vec4 fragColor;
```

Dopodiché cercheremo tutte le ricorrenze di "fragCoord" e le sostituiremo con "gl_FragCoord".

[11] https://www.shadertoy.com/user/FabriceNeyret2

Uniform in Ingresso

Shadertoy contiene una lista di variabili Uniform pre-esistenti, consultabile sul sito in cima alla finestra con il codice, facendo click su una freccia indicata come “Shader Inputs”, oppure facendo click sul “?” in fondo a destra della stessa finestra per aprire una finestra a pop-up contenente gli “Shadertoy Inputs” insieme ad altre informazioni. Passeremo in rassegna i principali Sampler e Uniform associati agli shader di Shadertoy.

Sampler

Shadertoy chiama i suoi Sampler in ingresso “iChannels”. Questi Sampler possono essere immagini, video, pattern casuali, mappe, eccetera. Il TOP “GLSL” ha una variabile simile chiamata “sTD2DInputs”. I Sampler di Shadertoy sono numerati individualmente, come “iChannel0”, “iChannel1” e così via, mentre in TouchDesigner “sTD2DInputs” è un vettore, quindi è possibile accedere ai suoi elementi con un indice numerico.

Ora, cerchiamo nel codice tutti i riferimenti a “iChannel0”, sostituendoli con “sTD2DInputs[0]”, e ogni volta che troviamo un riferimento a “iChannel1” sostituiamolo con “sTD2DInputs[1]”.

iGlobalTime

Per scoprire di che tipo di Uniform si tratti, osserviamo la lista “Shader Inputs” su Shadertoy menzionata in precedenza . All’interno della lista, “iGlobalTime” risulta come una variabile decimale, quindi la dichiarazione vicino alla cima del nostro codice, subito sotto quella di “fragColor”, dovrà essere:

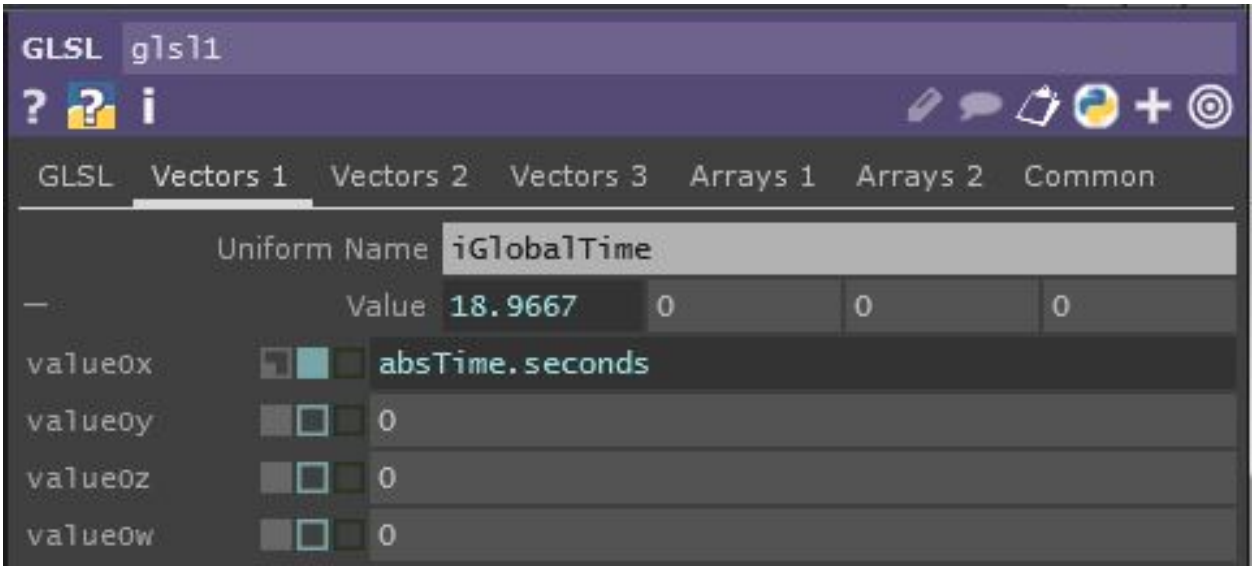
```
uniform float iGlobalTime;
```

Successivamente facciamo click sul TOP “GLSL” in TouchDesigner e andiamo nella pagina dei parametri “Vectors 1”.

Come prima “Uniform Name” scriveremo “iGlobalTime”, e come valore inseriremo

```
absTime.seconds
```

Dovrebbe apparire così:



iGlobalTime : absTime.seconds

iResolution

iResolution è la risoluzione dello shader su Shadertoy. Se la nostra risoluzione dipende da uno dei nostri ingressi, possiamo usare il vettore di TouchDesigner:

```
uTD2DInfos[i].res
```

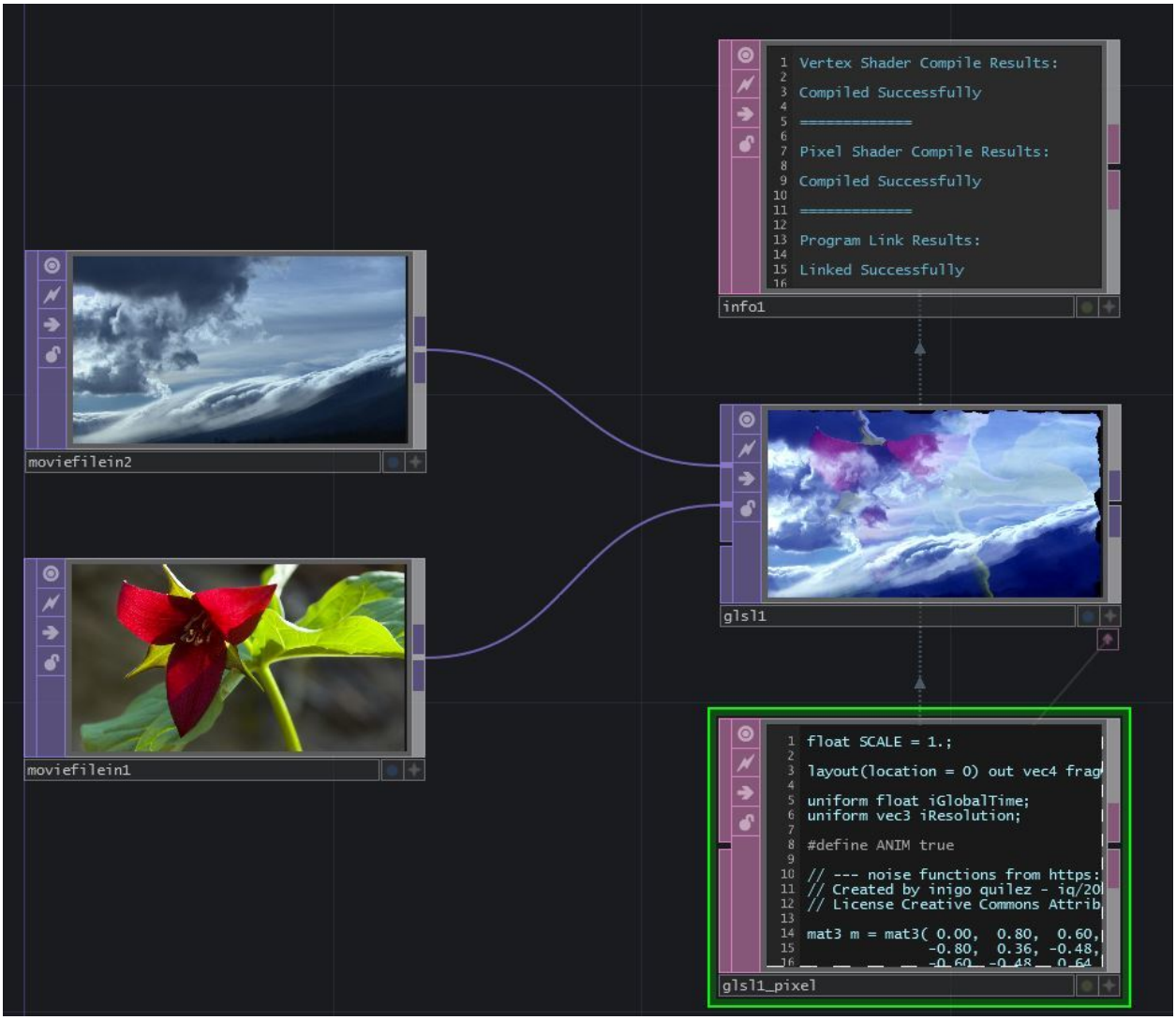
In questo caso possiamo aggiungere ".zw" alla fine per ottenere la larghezza ed altezza in pixel, oppure possiamo inserire ".xy" per ricevere il risultato di 1.0 diviso per la lunghezza e di 1.0 diviso per la larghezza.

Per questo esempio dichiareremo manualmente iResolution come Uniform. Se guardiamo la lista degli input di Shadertoy vedremo che iResolution è un vec3. In modo simile a iGlobalTime, inizieremo dichiarandola nel codice, inserendo la riga:

```
uniform vec3 iResolution;
```

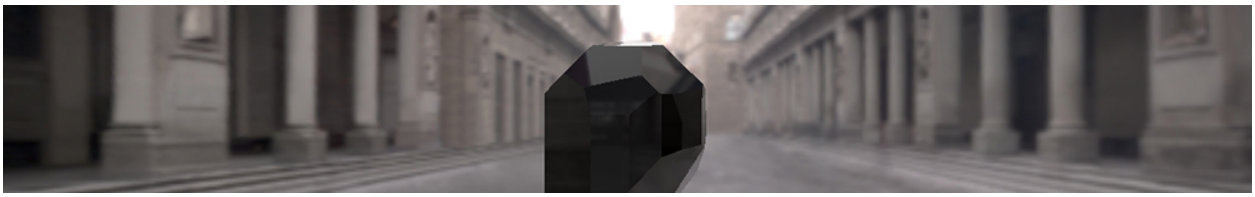
Dopodiché andremo nella pagina "Vectors 1" dei parametri del TOP "GLSL", e nel secondo "Uniform Name" inseriremo "iResolution". Come suoi valori scriviamo "1270" e "720". Non avremo bisogno del terzo valore del vec3, quindi lasceremo gli altri valori a "0".

Il nostro TOP "GLSL" dovrebbe essere ora in grado di venire compilato, e dovrebbe assomigliare a questo:



Example 1 compiled

Esempio 2: Shard



Example 2: Shard

Shader scritto da: simesgreen [12]

https://www.shadertoy.com/view/Xdf3zM

[12] https://www.shadertoy.com/user/simesgreen

Questo esempio ci porterà un po' più avanti, utilizzando cubemaps, creando Sampler casuali, sfruttando il suono e aggiungendo l'interazione con il mouse.

Setup

Inizieremo con un nuovo progetto di TouchDesigner e cominceremo nello stesso modo dello scorso esempio.

Creiamo un TOP "GLSL" e impostiamo la sua "Output Resolution" a "1280" e "720".

Creiamo quindi un DAT "Info" e nel suo parametro "Operator" facciamo riferimento al TOP "GLSL".

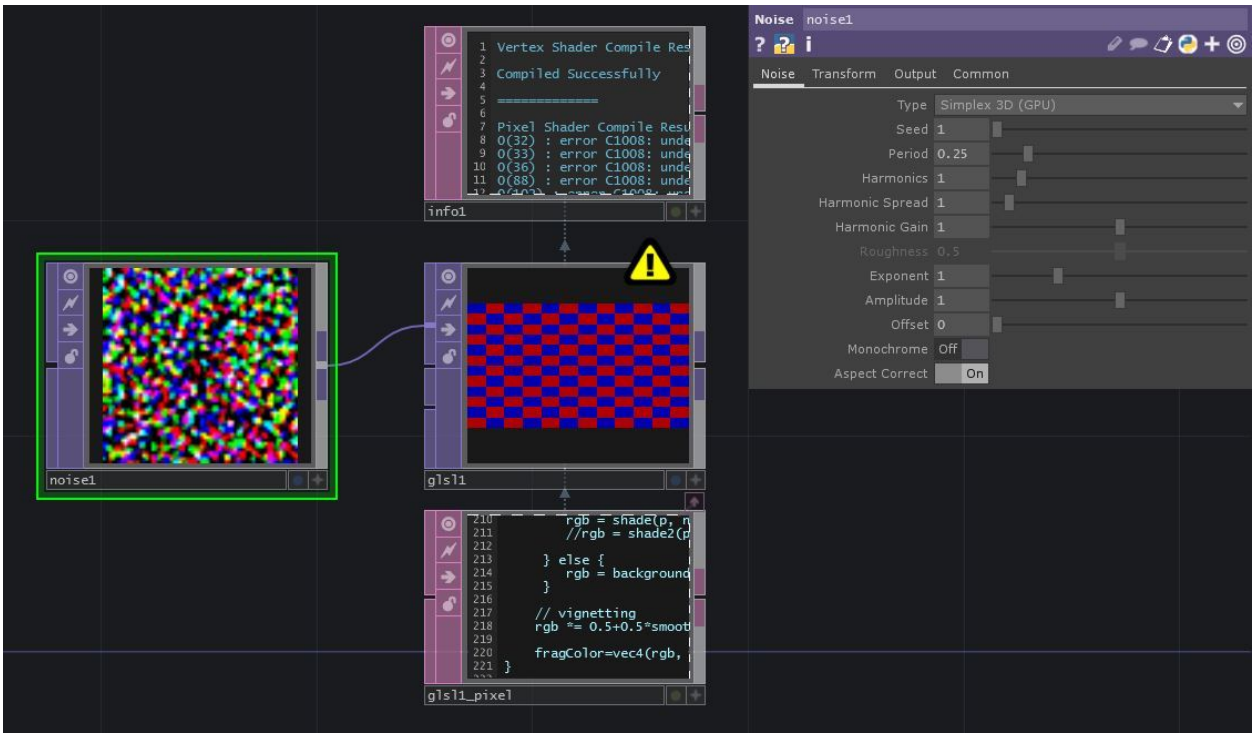
Copiamo ora il codice da Shadertoy in "glsl1_pixel".

Se osserviamo lo shader sul sito di Shadertoy, in fondo possiamo osservare che esso richiedere tre ingressi: una texture con valori casuali, un'immagine di sfondo e dell'audio.

Texture

In Shadertoy esistono quattro texture con valori casuali: una monocromatica e una a colori con risoluzione di 64x64 pixel, una monocromatica e una a colori con risoluzione 256x256 pixel.

Per questo esempio creiamo un TOP "Noise" e impostiamo la risoluzione a 64x64 pixel dalla pagina "Common" dei suoi parametri. Possiamo osservare la texture su Shadertoy e stimarne le impostazioni utilizzate. Questi sono i valori che possiamo utilizzare per adesso:



Ex2: Noise

Immagine di sfondo

Se facciamo click su “iChannel1” in Shadertoy, vedremo che è una texture dalla sezione “Cubemaps”. Di seguito è riportato un link per la fonte:

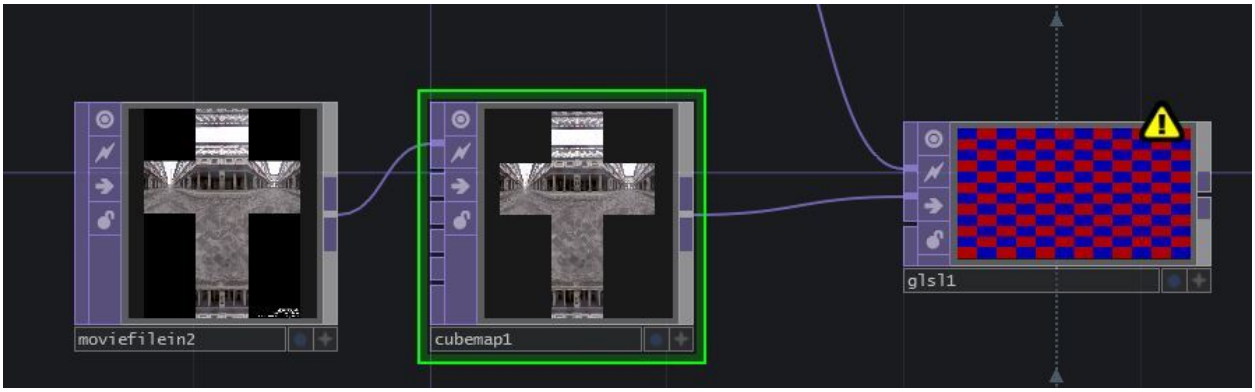
http://www.pauldebevec.com/Probes

Vicino alla fine della pagina sono presenti delle immagini formato-cubo che possono essere scaricate. L’immagine “LDR.tif” che coincide con quella utilizzata nello shader di Shadertoy è in fondo a destra, chiamata “Uffizi.tif”. Scarichiamo quest’immagine.

Creiamo un TOP “Movie File In” in TouchDesigner e facciamo riferimento al file scaricato.

Colleghiamo ora il nuovo TOP “Movie File In” ad un TOP “Cube Map”.

Impostiamo “Input Layout” su “Vertical Cross” e colleghiamo il TOP “Cube Map” al secondo ingresso di “glsl1”.



Ex2: Cube Map

Audio

“iChannel2” in Shadertoy è collegato ad un input da Soundcloud, ma all’interno di TouchDesigner possiamo usare qualsiasi audio preferiamo; per adesso ci limiteremo alla traccia di default presente nel CHOP “Audio File In”.

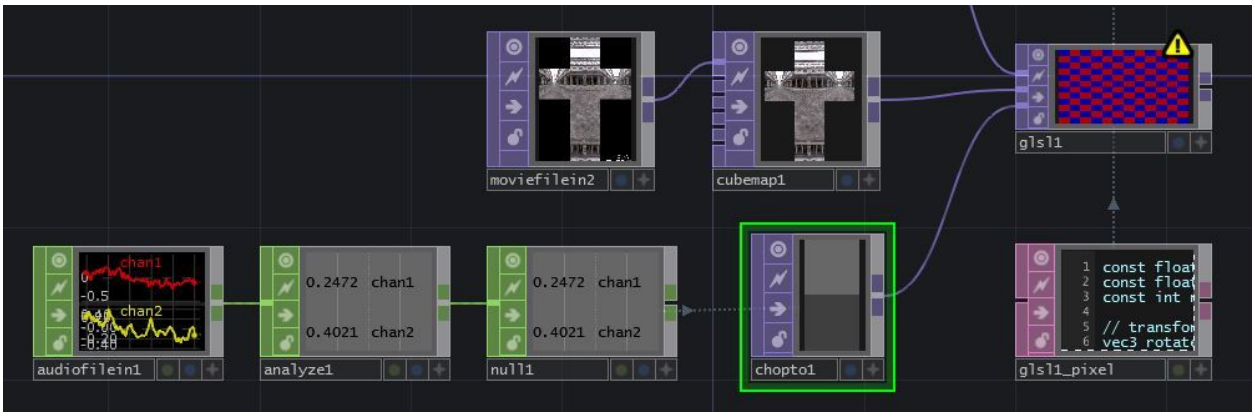
Abbiamo bisogno di preparare l’audio per il TOP “GLSL”, dato che non possiamo collegare direttamente un CHOP “Audio” ad un TOP “GLSL”. La riga 35 del codice GLSL è commentata, e dice che possiamo usare questa linea di codice se non abbiamo un file audio, ma non è il nostro caso. Nella riga successiva la variabile “d” sta cercando una “texture”, con il commento “move planes with music”(anima i piani con la musica). Il modo più facile per ottenere una semplice versione di ciò consiste nel convertire il CHOP “Audio File In” in una texture.

Vogliamo solamente un campione per questo esempio, quindi aggiungiamo un CHOP “Analyze” dopo il CHOP “Audio File In” e impostiamo il parametro “Function” su “Maximum”; potremmo voler regolare questo valore più avanti, quindi inseriamo un CHOP “Null” dopo di questo.

Ora creiamo un TOP “CHOP to” che faccia riferimento al CHOP “Null” e impostiamo il “Data Format” su “R”.

Colleghiamo infine il TOP “CHOP to” al terzo ingresso del TOP “GLSL”.

Il network dovrebbe apparire così:



Ex2: Audio

Ora che gli ingressi sono impostati correttamente, possiamo dare un’occhiata al DAT “Info” e vedere cosa occorre cambiare in questo codice.

Funzione Principale e fragColor

Prima di tutto cambieremo il nome della funzione principale da “mainImage” a “main” e rimuoviamone i parametri, così da avere al posto di questo:

```
void mainImage( out vec4 fragColor, in vec2 fragCoord )
```

solamente questa espressione:

```
void main()
```

Ora dobbiamo rimpiazzare i due parametri che abbiamo rimosso dalla funzione principale: fragColor e fragCoord. Andiamo all’inizio del codice e inseriamo:

```
layout(location = 0) out vec4 fragColor;
```

Possiamo adesso cercare tutte le ricorrenze di fragCoord (dovrebbe essercene una sola, nella prima linea della funzione principale) e sostituiamole con la variabile “gl_FragCoord”.

Sampler

iChannel0

In modo simile all’ultimo esempio, iChannel0 è un ingresso 2D, quindi dobbiamo trovare tutti i riferimenti a “iChannel0” e sostituirli con “sTD2DInputs[0]”. In GLSL 3.30 e successivo non abbiamo bisogno di utilizzare una diversa funzione “texture()” per tipi di texture diversi, quindi usiamo lo

strumento “trova e sostituisci” per rimpiazzare tutte le “texture2d” con “texture” (“texture2D” può funzionare ancora con alcuni driver ma, per assicurarci la massima compatibilità del nostro shader su tutte le GPU e con ogni versione di driver, occorre compiere questa sostituzione.). Se lo shader sul sito di Shadertoy fosse stato nel frattempo aggiornato, potrebbe non essere necessario eseguire tale operazione.

```
float j = texelSize.x*iGlobalFrame;  // animate planes
vec3 s = texture(sTD2DInputs[0], vec2(float(i)*texelSize, j)).xyz*2.0-1.0;
vec3 n = normalize(s);
```

Ex2: texture

iChannel1

Modificare il codice per iChannel1 è simile a quanto fatto per iChannel0, ma l’input è una cubemap. TouchDesigner ordina già per noi il tipo di ingressi in vettori separati: 2D, 3D, cube, eccetera. Se questo fosse stato un altro ingresso 2D avremmo potuto utilizzare “sTD2DInputs[1]”, ma dato che è una cubemap (la prima e unica nei nostri ingressi) usiamo “sTDCubeInputs[0]”.

In modo simile a quanto fatto per “iChannel0”, se lo shader ha dei riferimenti a “textureCube”, dobbiamo modificarli in “texture”, ma se lo shader sul sito fosse stato nel frattempo aggiornato è possibile che quest’ultimo passaggio non sia necessario.

iChannel2

“iChannel2” è l’audio. Abbiamo convertito le informazioni dell’audio in una texture 2D con il TOP “CHOP to”; questa texture è il terzo ingresso, ma è il secondo input in 2D, quindi dobbiamo rimpiazzare tutti gli “iChannel2” con “sTD2DInputs[1]”.

Se salviamo lo shader e controlliamo il DAT “Info”, dovremmo vedere corretti tutti gli errori che erano presenti prima, a parte quelli relativi ad “iMouse”.

iMouse

Per scoprire di cosa abbiamo bisogno per imitare lo Uniform “iMouse” di Shadertoy, dobbiamo tornare alla lista degli ingressi dello shader. Facciamo click sul punto di domanda in fondo a destra nella finestra, e scendendo troveremo “iMouse”:

vec3	iChannelResolution[4]	image/sound	Input texture resolution for each channel
vec4	iMouse	image	xy = current pixel coords (if LMB is down). zw = click pixel
sampler2D	iChannel{i}	image/sound	Sampler for input textures i

Ex2: iMouse Definition

“iMouse” è un “vec4” con 4 valori definiti come:

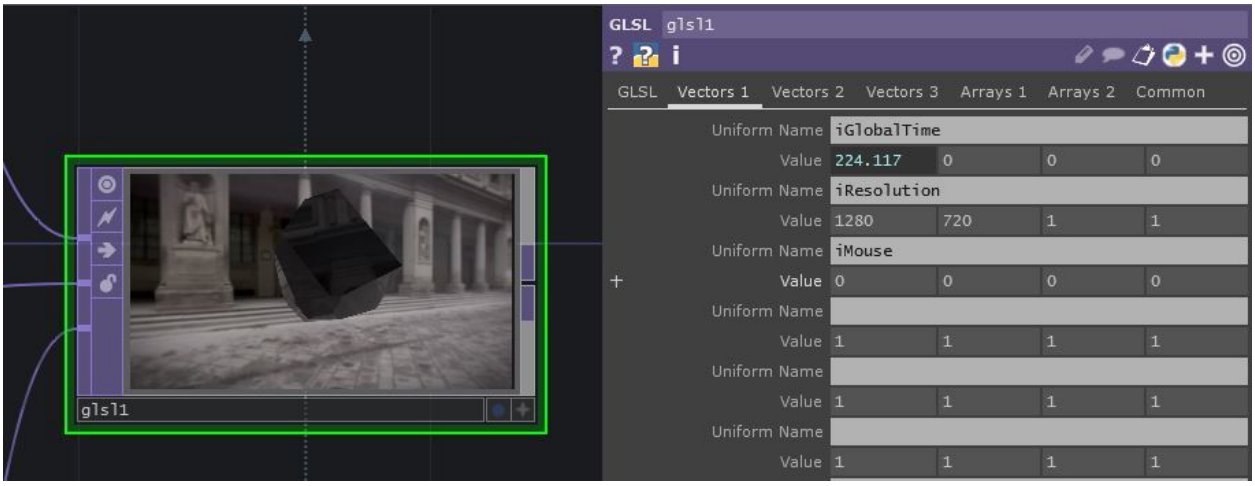
```
xy = current pixel coords (if LMB is down). zw = click pixel
```

Per adesso limitiamoci a creare la Uniform con il minimo necessario perché lo shader renderizzi senza errori. Aggiungiamo “iMouse” come “Uniform Name” alla pagina “Vectors 1” dei parametri del TOP “GLSL”, e impostiamo il suo valore a “0”.
 Ora aggiungiamo:

```
uniform vec4 iMouse;
```

dopo le altre dichiarazioni delle Uniform vicino all'inizio del codice.

Il TOP "GLSL" dovrebbe adesso risultare privo di errori, e dovrebbe assomigliare a questo:



Ex2: iMouse success

Funzionalità di iMouse

Torniamo alla descrizione di ciò che "iMouse" fa:

```
xy = current pixel coords (if LMB is down). zw = click pixel
```

Possiamo anche osservare un esempio di iMouse in azione in questo shader:

https://www.shadertoy.com/view/Mss3zH.

Dunque ".xy" sono le coordinate del mouse quando il tasto sinistro è premuto, mentre ".zw" sono le coordinate di dove si trovava inizialmente il mouse quando ha fatto click. Dobbiamo organizzare un piccolo network di CHOP per ricreare questi dati.

Potremmo usare il CHOP "Mouse In", ma è preferibile che il mouse interagisca con lo shader solo quando è esplicitamente richiesto.

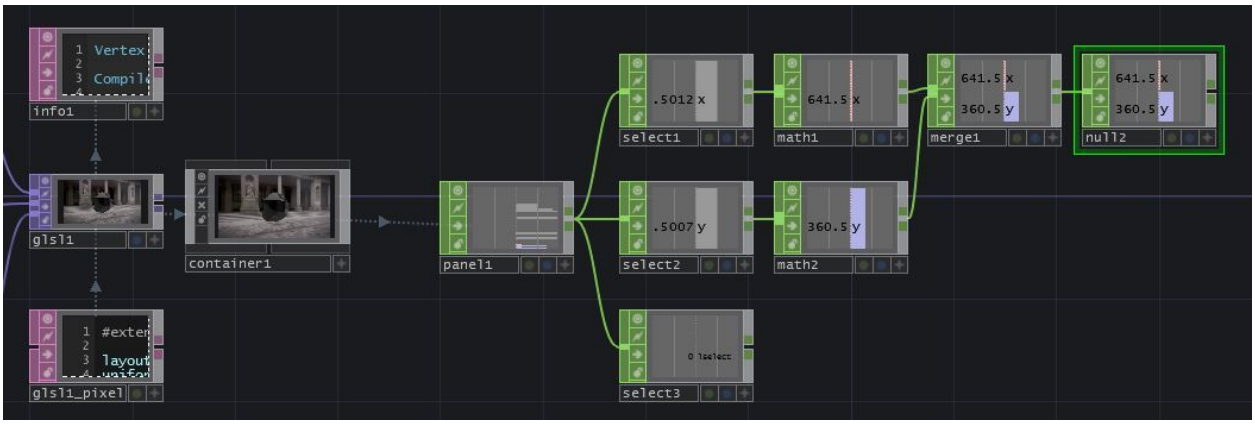
Creiamo un COMP "Container", impostiamo "Width" e "Height" sulle stesse dimensioni del TOP "GLSL", in questo caso "1280" e "720". Impostiamo il TOP "GLSL" come "Background" del container dal pannello delle impostazioni. Ora creiamo un CHOP "Panel" e mettiamo un riferimento al nostro container nel parametro "Component".

Vogliamo selezionare tre valori dal CHOP "Panel": "u", "v" e "lselect". Per fare ciò aggiungiamo tre CHOP "Select" e colleghiamo ognuno di essi a "panel1" in parallelo.

All'interno di "select1", nel campo "Channel Names", inseriamo "u". Lo rinomineremo subito, quindi nel campo "Rename From" inseriamo "u" e in "Rename To" scriviamo "x". In "select2" selezioniamo il canale "v" e, come prima, rinominiamo da "v" a "y". Per "select3" non abbiamo bisogno di rinominare nulla, quindi limitiamoci ad inserire "lselect" nel campo "Channel Names".

Avremo bisogno di convertire questi valori dal range “0-1” all’intera risoluzione. Aggiungiamo un CHOP “Math” dopo “select1” e impostiamo il parametro “Multiply” a “1280”. Aggiungiamo un CHOP “Math” dopo “select2” e impostiamo il parametro “Multiply” a “720”.

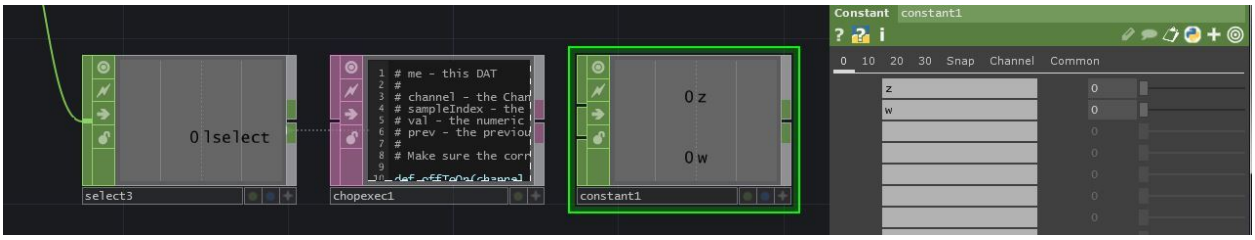
Ora creiamo un CHOP “Merge” e colleghiamo ad esso “math1” e “math2”. Come sempre aggiungiamo un CHOP “Null” dopo di questo. Per adesso dovrebbe apparire così:



Ex2: iMouse Network

Ora abbiamo i nostri valori “.xy”, quindi ci restano da settare i valori “.zw”. Per fare ciò abbiamo bisogno di leggere la posizione del mouse quando ha appena fatto click, e di mantenere in memoria quel valore finché non viene rilasciato il tasto del mouse.

Creiamo un DAT “CHOP Execute” e inseriamo “select 3” nel campo CHOP. Assicuriamoci che i tasti “Off to On” e “On to Off” siano su “on” e tutti gli altri siano su “off”. Aggiungiamo un CHOP “Constant” al network e nei primi due campi “Name” creiamo i canali “z” e “w”. Dovrebbe apparire così:



Ex2: iMouse Constant CHOP

Quando il mouse viene premuto vogliamo scrivere i valori di “null2” sui canali “z” e “w” di “constant1”, e vogliamo che questi tornino a “0” quando il tasto è rilasciato.

Aggiungiamo il seguente codice alla funzione “offToOn”:

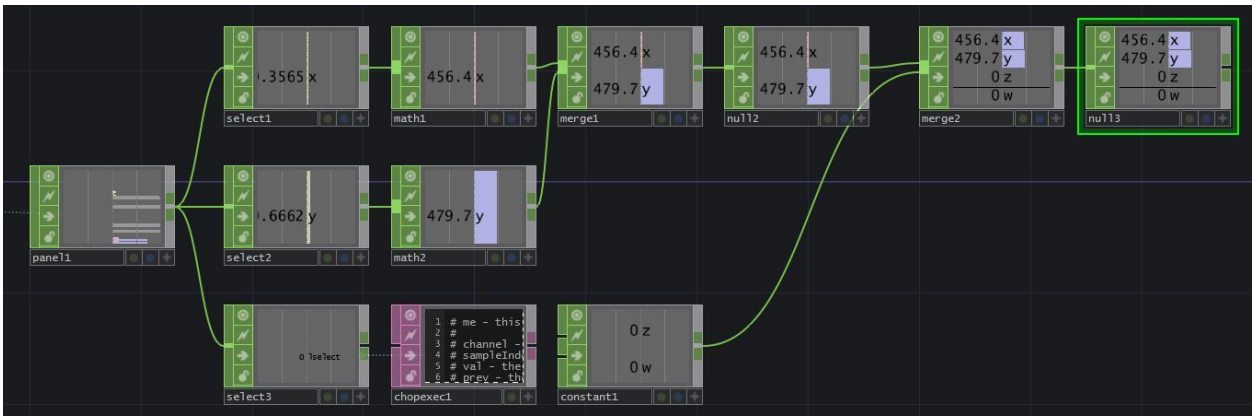
```
z = op('null2')['x']
w = op('null2')['y']
op('constant1').par.value0 = z
op('constant1').par.value1 = w
```

e scriviamo questo nella sezione “onToOff”

```
op('constant1').par.value0 = 0
op('constant1').par.value1 = 0
```

Possiamo usare un altro CHOP “Merge” per unire “constant1” con “null2” e aggiungere un altro CHOP “Null” dopo di questo. Torniamo al TOP “GLSL” e nei quattro campi “Value” della Uniform “iMouse” possiamo inserire i riferimenti ai canali “x”, “y”, “z” e “w” di “null3”. Possiamo fare ciò con degli export o con dei riferimenti in Python.

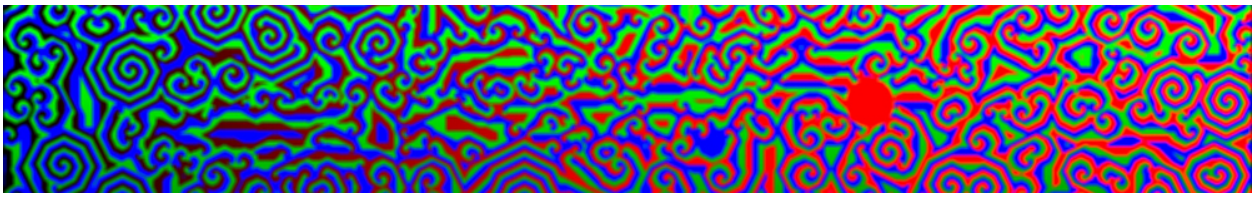
La pagina “Vectors 1” dovrebbe apparire così:



Ex2: Final iMouse Network

Se guardiamo il container dovremmo ora essere in grado di fare click e trascinare per ruotare attorno allo Shard!

Esempio 3: Cyclic Cellular Automaton



Ex2: Example 3: Cyclic Cellular Automaton

shader scritto da: zolozulman[13]

[13] https://www.shadertoy.com/user/zolozulman

https://www.shadertoy.com/view/4tV3Wd

Shadertoy implementa l'utilizzo di buffer multipli, separando le funzioni in processi diversi. L'esempio di seguito mostra un modo per importare questi shader multi-pass.

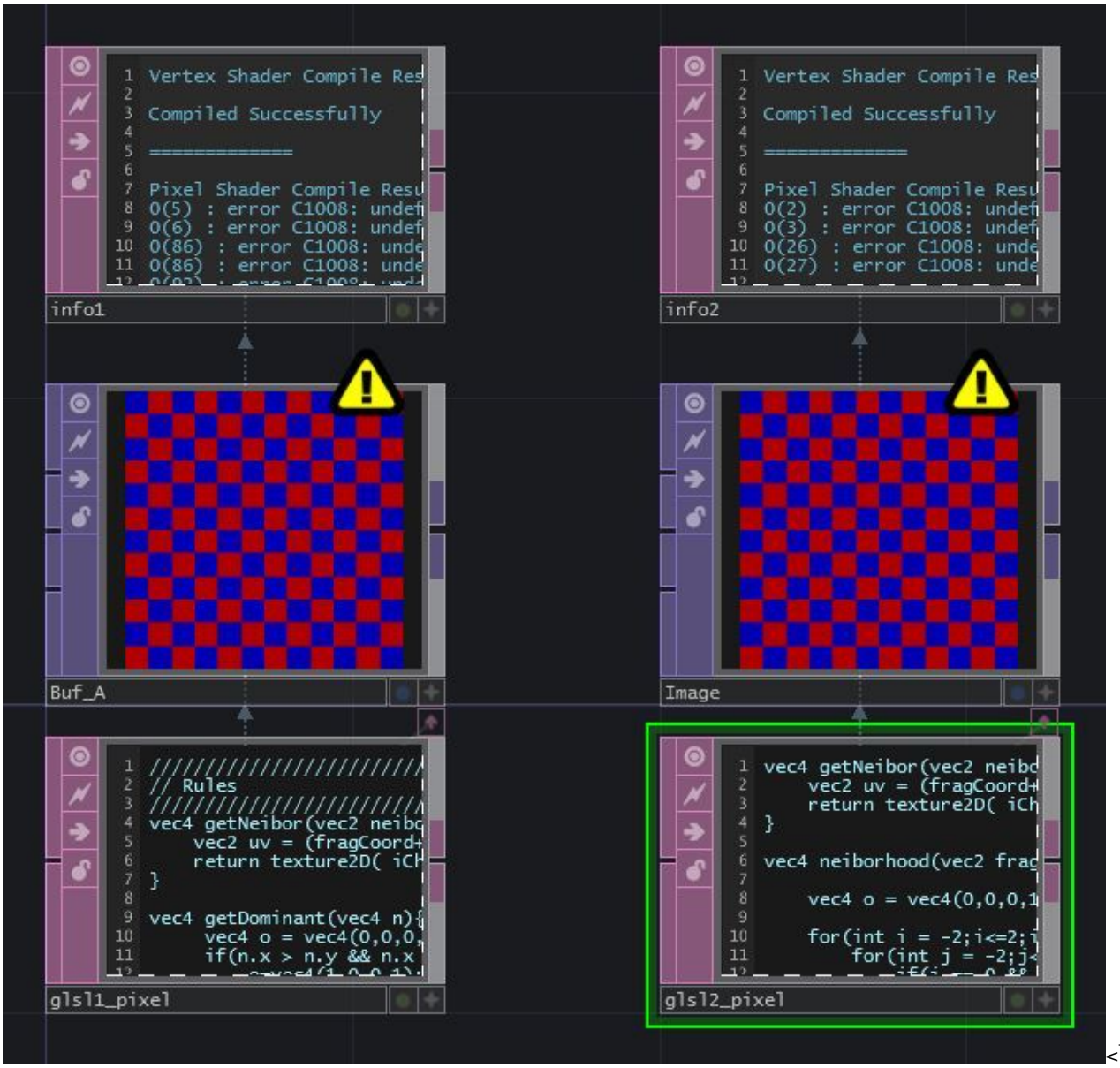
Setup

Collegamento dei Buffer

Sul sito Shadertoy l'esempio precedente aveva una sola finestra, chiamata "Image", che conteneva tutto il codice. Questo esempio ha invece una sezione chiamata "Image" e una chiamata "Buf_A": ciò significa che dovremo utilizzare due TOP "GLSL" differenti per rappresentare ognuna delle funzioni dello shader, dette "buffers".

Iniziamo a creare questi due TOP "GLSL", e impostiamoli entrambi con una risoluzione di 1280x720 pixel. Creiamo inoltre un DAT "Info" per ognuno di essi. Rinominiamo quindi ognuno dei TOP "GLSL" per rispecchiare i buffer, così che possiamo capire meglio le corrispondenze.

Ora possiamo copiare il codice da ognuno dei buffer e incollarlo nel pixel shader "GLSL" corrispondente.
 Dovrebbe apparire così:

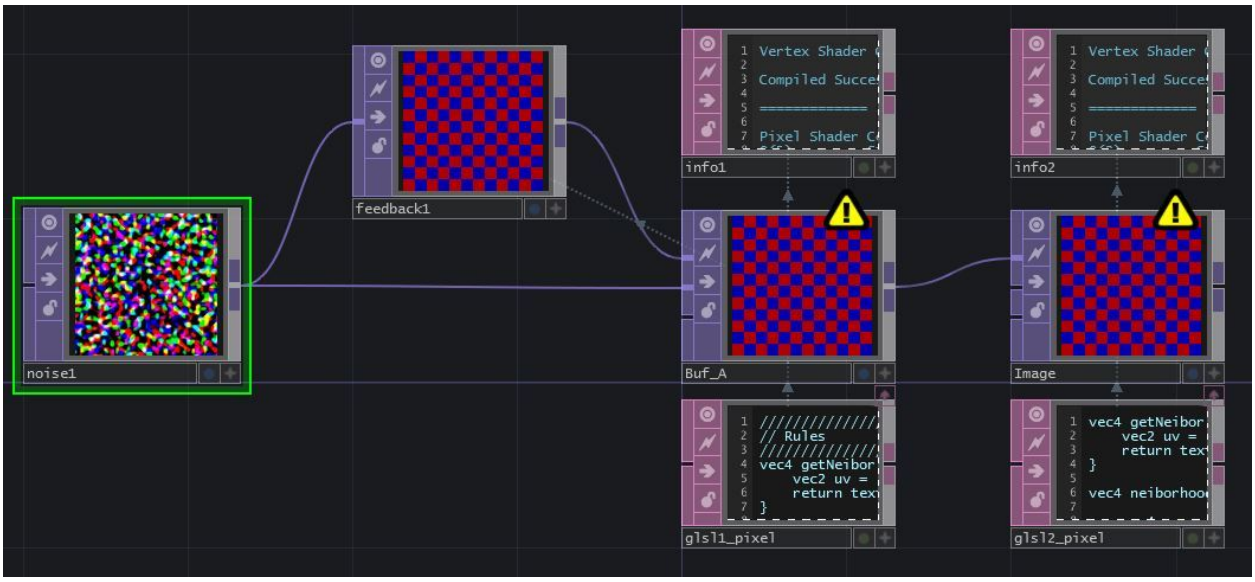


TOP Noise e Feedback

Per il buffer “Image”, “iChannel0” corrisponde a “Buf_A”. Ciò significa che possiamo collegare l’uscita del nostro TOP “GLSL” chiamato “Buf_A” al primo ingresso del nostro TOP “GLSL” con nome “Image”. Se facciamo click sulla sezione “Buf_A” in Shadertoy possiamo vedere che “iChannel0” è un feedback. Prima di creare questo loop di feedback, lavoriamo con “iChannel1”: questo ingresso è una texture di valori casuali, quindi possiamo creare un TOP “Noise” con le stesse impostazioni dell’esempio precedente, e collegarlo al secondo ingresso del TOP “GLSL” chiamato “Buf_A”.

Per il loop di feedback non possiamo collegare l'uscita di un TOP all'ingresso dello stesso senza incappare in un errore, aggiungiamo perciò un TOP "Feedback" al network. Il TOP "Feedback" ha bisogno di un input, quindi possiamo collegargli il TOP "Noise" in ingresso, impostare il parametro "Target TOP" a "Buf_A" e infine collegare la sua uscita al primo ingresso del TOP "GLSL" "Buf_A".

Il nostro network dovrebbe apparire così:



Ex2: Example 3: Noise and Feedback

Funzione Principale e fragColor

Affrontiamo lo stesso processo dell'esempio precedente, cambiando "mainImage" in "main", rimuovendo i parametri all'interno delle parentesi e dichiarando all'inizio del codice:

```
layout(location = 0) out vec4 fragColor;
```

Dopodiché sostituiamo in entrambi gli shader tutte le ricorrenze di "fragCoord" con "gl_FragCoord". Se osserviamo i DAT "Info" possiamo notare un errore relativo al tipo di dati: ciò è dovuto al fatto che la funzione principale richieda "vec2 fragCoord", ma la variabile "gl_FragCoord" è un vec4.

Avremo bisogno di passare in rassegna la funzione principale e, ogni volta che troviamo la variabile "gl_FragCoord" senza ".x" o ".y" dopo di essa, dovremo aggiungere ".xy" (se lo cambiamo nella funzione principale sarà trasmesso come vec2 anche alle altre funzioni, che è ciò che vogliamo.). Ricordate che se il codice fa riferimento a "gl_FragCoord.x" o a "gl_FragCoord.y" non è necessario apportare modifiche, poiché ".x" e ".y" stanno già selezionando un singolo valore all'interno del vettore.

Questo esempio ha una sola ricorrenza in ognuna delle funzioni principali che necessita di essere cambiata. La funzione principale di "Buf_A" dovrebbe apparire così:

```
void main()
{
    float r = pow(iMouse.x-gl_FragCoord.x,2.0) + pow(iMouse.y-gl_FragCoord.y,2.0);
    float max_r = pow(25.0,2.0);
    vec4 c = vec4(0,0,0,1);
    if(r <= max_r){
        c = vec4(1,0,0,1);
    }
    vec2 uv = gl_FragCoord.xy / iResolution.xy;
    if(iFrame < 5){
        fragColor = texture2D( iChannel1, uv);
    }else{
        fragColor = c+rule(gl_FragCoord.xy);
    }
}
```

mentre la funzione principale di “Image” dovrebbe essere così:

```
void main()
{
    vec2 uv = gl_FragCoord.xy / iResolution.xy;
    fragColor = neiborhood(gl_FragCoord.xy)*vec4(uv,0.5+0.5*sin(iGlobalTime),1.0)*4.;
}
```

Ora possiamo tornare al DAT “Info” e vedere cos’altro occorre cambiare.

iResolution, iGlobalTime, iMouse, e iFrame

Entrambi “Buf_A” e “Image” richiedono la dichiarazione di “iResolution” e “iGlobalTime”, che abbiamo già fatto in precedenza, quindi procederemo ad aggiungerli. Abbiamo bisogno di: ‘ uniform vec3 Resolution;

uniform float iGlobalTime; ‘

da inserire all’inizio di entrambi i pixel shader, ricordandoci poi di aggiungere entrambe le Uniform alla pagina “Vectors 1” dei parametri del TOP “GLSL”.

Se osserviamo il DAT “Info” relativo a “Buf_A”, possiamo vedere una nuova variabile indefinita: “iFrame”. Questa Uniform è un contatore di fotogrammi e possiamo fare riferimento a “travestimento” o “me.time.frame”, a seconda se vogliamo che il contatore vada in loop con la timeline oppure no. Per questo esempio usiamo “absTime.frame” come espressione per il primo valore della Uniform, perché non vogliamo che il loop della timeline influenzi lo shader.

Possiamo quindi copiare e incollare lo stesso network creato per “iMouse” nell’esempio precedente e dichiarare la Uniform nello stesso modo.

iChannels

L'unica cosa rimasta da convertire sono i riferimenti di "iChannel0", da cambiare in "sTD2DInputs[0]", e di "iChannel1" da sostituire con "sTD2DInputs[1]". Possiamo farlo per entrambi i pixel shader.

Entrambi i nostri TOP "GLSL" adesso dovrebbero funzionare, e il network dovrebbe assomigliare a questo:

Ex2: Example 3: Noise and Feedback

12.7 Sistemi di Particelle sulla GPU

12.7.1 Introduzione

Questo tutorial si concentrerà sulla creazione di un sistema di particelle basato sulla GPU, utilizzando TouchDesigner e gli shader GLSL. Ciò comporta molti benefici rispetto ad usare il già presente SOP "Particle", poiché tutti i calcoli all'interno dello shader GLSL verranno compiuti dalla scheda grafica; in questo modo si riuscirà a creare sistemi di particelle con moltissimi elementi e con comportamenti personalizzati, ma che vengano comunque calcolati velocemente.

Il processo per ottenere un sistema di particelle in GLSL all'inizio può risultare leggermente complicato rispetto al semplice utilizzo del SOP "Particle" ma, una volta capiti i principi basilari che si celano dietro al flusso di lavoro, avrete la totale libertà di sperimentare.

Un importante concetto da assimilare è che lo shader GLSL che utilizzeremo non può creare punti dal nulla, quindi occorrerà fornirgli un insieme di punti di partenza corrispondente al numero massimo di particelle presenti nel sistema. La quantità di punti determinerà il numero di volte in cui il nostro shader verrà computato e in cui il codice che scriveremo sarà applicato individualmente ad ogni particella.

Un secondo concetto da ricordare è che, in modo simile ad ogni altro tipo di shader, il codice non fa mai riferimento a nient'altro che il punto che sta correntemente processando. Lo shader non avrà riferimenti riguardo i fotogrammi precedenti o successivi, e non avrà nessun collegamento con gli altri punti calcolati nello stesso frame. Tutto il codice dello shader dovrà essere autosufficiente e ogni dato di riferimento dovrà essergli fornito come una texture o come un valore "uniform". Un "uniform" è un parametro o un valore che potete trasmettere al codice dello shader, importante perché le particelle in un sistema di particelle hanno bisogno di conoscere la loro ultima posizione calcolata, così che possano riutilizzare il codice per calcolare la loro nuova posizione nel frame successivo. In altre applicazioni GLSL avreste bisogno di creare diversi buffer delle texture, da far rimbalzare tra lettura e scrittura; in TouchDesigner si può invece risolvere utilizzando un semplice sistema di feedback.

Per raggiungere l'obiettivo di creare un sistema di particelle pienamente funzionante e basato sulla GPU, divideremo il processo in un numero incrementale di passaggi successivi.

12.7.2 Spostare le Particelle con le Texture

Prima di iniziare ecco un elenco dei vari passaggi che affronteremo in questa sezione:

1. Creare un insieme di punti che rappresenti le posizioni iniziali;
2. Assegnare un indice ad ogni punto;
3. Usare un TOP "Noise" per creare sufficienti canali RGBA da poter usare come posizioni dei punti;
4. Creare un TOP "GLSL" per scalare i valori del TOP "Noise" (sarà modificato dopo che avremo rimosso il TOP "Noise");

5. Creare un setup di render basilare;
6. Creare un MAT “GLSL” che contenga tutto il codice dello shader relativo alle particelle;
7. Creare uno shader Pixel per rendere bianchi tutti i pixel;
8. Creare uno shader Vertex con ingressi e output corretti;
9. Trovare il “pointIndex”, la posizione delle texture dei punti e i punti per ogni istanza;
10. Campionare la texture del TOP “Noise” e creare delle coordinare UV;
11. Muovere i punti sfruttando i passaggi precedenti.

La prima cosa che dobbiamo fare è creare un insieme di punti che possano essere usati dal MAT “GLSL” come particelle. Come detto in precedenza, un MAT “GLSL” non può creare particelle dal nulla, esso è semplicemente uno script che viene applicato ad ogni punto che gli venga dato. Per il sistema di particelle questo significa che abbiamo bisogno di una geometria con mille punti per avere un sistema di particelle con mille punti. In modo simile, se volessimo un sistema di particelle con un milione di punti, avremmo bisogno di una geometria con un milione di punti. Per iniziare in modo semplice, genereremo mille punti nell’origine (0,0,0).

Potete seguire la spiegazione successiva aprendo l’esempio “01_Moving_particles_with_texture.toe”.

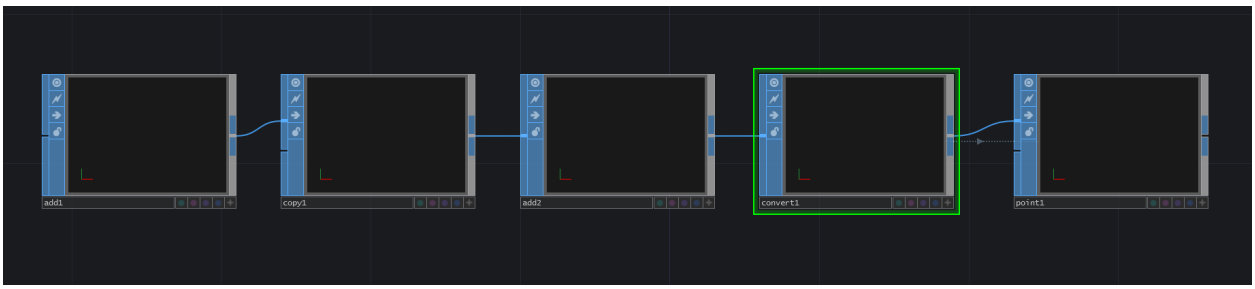
Iniziamo creando un SOP “Add” e facendo click sulla prima casella nella finestra dei parametri per abilitare “Point 0”. I valori di default di “Point 0” andranno bene per questo esempio. Abbiamo creato il nostro primo punto. Ora connettiamo l’uscita del SOP “Add” al primo ingresso di un SOP “Copy”. Il SOP “Copy” è utilizzato per copiare il primo punto creato dal SOP “Add”, restituendo più punti per il nostro sistema di particelle. Fate ciò cambiando il parametro “Number of Copies” del SOP “Copy” in 1000. Abbiamo creato 1000 punti per poter generare un sistema di particelle con 1000 punti.

Colleghiamo l’output del SOP “Copy” al primo ingresso di un altro SOP “Add”, che verrà utilizzato per unire i nostri 1000 punti creando un poligono che possa essere convertito in particelle prima di entrare nello shader. Per fare ciò, nel nuovo SOP “Add” (che dovrebbe chiamarsi “add2” se state usando la nomenclatura di defalut degli Operatori) andate nella sezione “Polygon” nella finestra dei parametri, e aggiungete un asterisco (“*”) al primo parametro chiamato “Polygon”. Ora convertiremo questo poligono con 1000 punti in particelle. Fate ciò collegando l’uscita del secondo SOP “Add” (“add2”) ad un SOP “Convert”. Nel nuovo SOP “Convert” cambiamo il parametro chiamato “Convert to” in “Particles” e cambiamo il parametro chiamato “Particel Type” in “Render as Point Sprites”. Creare dei point sprite ci permetterà più avanti di utilizzare una singola linea di codice per aumentare la dimensione delle particelle.

L’ultimo passo prima di ottenere il nostro materiale è creare un attributo personalizzato aggiuntivo usando gli indici dei punti. Per fare ciò colleghiamo l’uscita del SOP “Convert” ad un SOP “Point”. Impostiamo il nome del primo “Custom Attrib” a “pointIndex”. Dal menu a tendina alla destra del campo con il nome selezioniamo “float” come tipo di dati. Espandiamo il campo “Value” e nel campo del primo parametro chiamato “custom1val1” inseriamo il seguente script Python:

```
1 me.inputPoint.index
```

Questo script crea per ogni punto un attributo personalizzato che possiamo utilizzare nel codice GLSL. In questo caso abbiamo preso l'indice di ogni punto e l'abbiamo assegnato ad un valore "float" chiamato "pointIndex". Ora abbiamo completato i primi due passaggi e abbiamo 1000 particelle che daremo in pasto al MAT "GLSL". Dovrebbe apparire così (nota: a questo punto non vedrete ancora nulla nei viewer dei vostri SOP, a meno che non abbiate abilitato qualche opzione di visualizzazione per rendere visibili i punti!):



La prossima cosa che faremo è creare dei valori casuali da usare come posizioni dei punti. Il primo passo è creare un TOP "Noise". Nei suoi parametri "Common" cambiate la risoluzione a 1000x1 pixel ed il "Pixel Format" a "32-bit float (RGBA)". Questo ci restituisce un pixel del TOP "Noise" per ogni particella che abbiamo (1000 pixel per 1000 particelle). Cambiare il formato a "32-bit float (RGBA)" implica che ogni pixel avrà 32 bit per ogni canale, comportando una precisione decisamente maggiore per i dati dei vari canali colore. Il passo successivo è impostare il parametro "Monochrome" ad "off". Questo restituisce valori casuali diversi per ogni canale colore, che saranno quindi tradotti in diversi valori casuali per le posizioni X, Y e Z delle nostre particelle. Possiamo quindi scegliere di animare il TOP "Noise" come vogliamo per questo esempio, ma la cosa più semplice è aggiungere il seguente codice al valore "tx" del parametro "Translate":

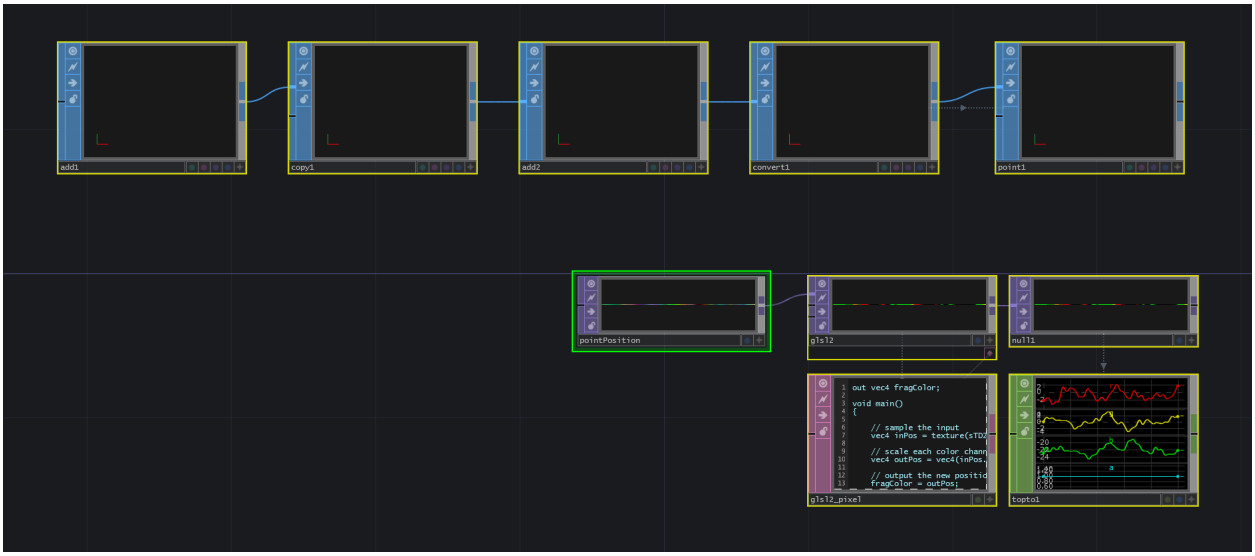
```
1  absTime.frame/100
```

Ciò trasformerà il TOP "Noise" lungo l'asse X, creando l'effetto di un nastro sul sistema iniziale di particelle. Ora creeremo un TOP "GLSL" che ci permetterà di avere un controllo più preciso sui valori correnti di ogni canale colore del TOP "Noise". Saremo in grado di scalare quei valori e in sezioni più avanzate di espandere lo stesso shader aggiungendo più funzionalità. Colleghiamo l'uscita del TOP "Noise" al primo ingresso del TOP "GLSL". Il TOP "GLSL" è creato di default con un DAT "Text" collegato ad esso, contenente uno shader che emette il colore bianco. Modifichiamo il DAT "Text" cancellando il codice esistente, aggiungiamo quello riportato di seguito e salviamo:

```
out vec4 fragColor;

void main()
{
// sample the input
vec4 inPosition = texture(sTD2DInputs[0], vUV.st);

// scale each color channel (the XYZ of the vector) separately
vec4 outPosition = vec4(inPosition.x * 5.0 - 2.5, inPosition.y * 5.0 - 2.5, inPositi\
on.z * -5.0 - 20.0, 1.0);

// output the new position
fragColor = outPosition;
}
```

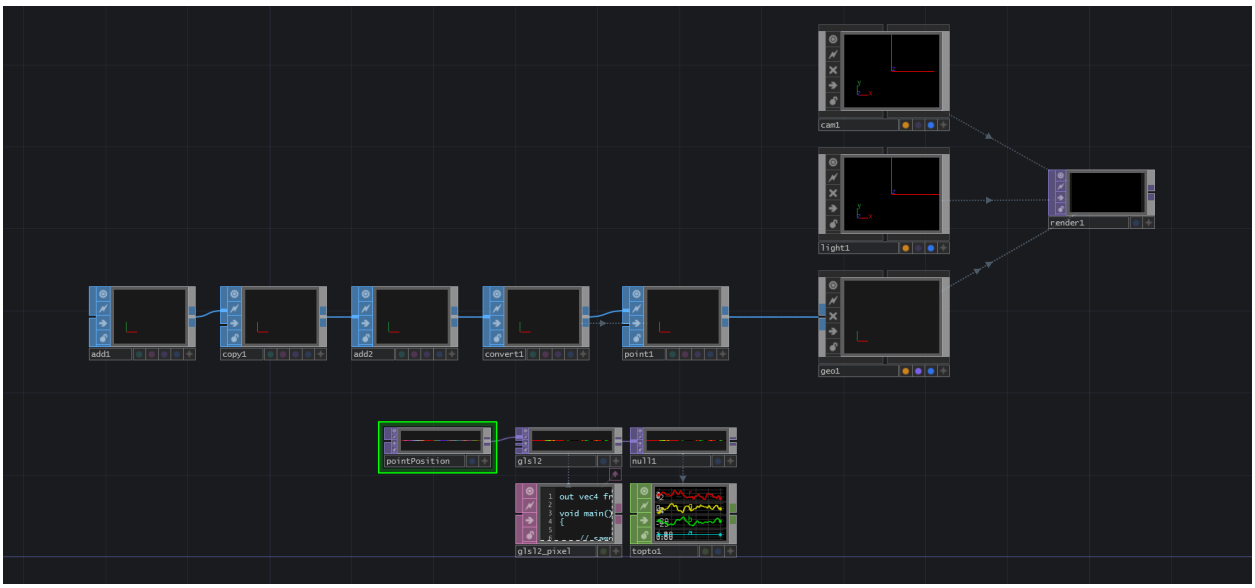
Analizzeremo velocemente il codice appena riportato, ma per ulteriori chierimenti fate riferimento alle sezioni precedenti di questo capitolo. Prima di tutto impostiamo l'uscita principale "fragColor", quindi campioniamo la texture alle UV correnti. Poiché abbiamo creato il TOP "Noise" per avere lo stesso numero di pixel delle particelle presenti, possiamo campionare un pixel per ogni particella, uno alla volta. Dopo aver campionato il pixel corrente scaliamo i canali R e G (le componenti X e Y del vec4) di 5.0 e quindi li trasliamo di 2.5 unità a sinistra e in basso rispetto alla telecamera. Scaliamo quindi il canale B (la Z del vec4) di -5.0 e lo trasliamo 20 unità allontanandolo dalla camera, così da mostrare nella scena tutto il sistema di particelle. Possiamo lasciare il canale alfa ad 1.0, poiché per il momento non lo utilizzeremo.

Dopo aver scalato e traslato, assegniamo la "outPosition" all'uscita "fragColor". Se volete vedere la posizioni che le particelle riceveranno potete connettere il TOP "GLSL" ad un Operatore "TOP to CHOP" e visualizzare i valori di ogni canale colore. Questo conclude i passi 3 e 4 della lista precedente.



Particle

Ora creiamo un semplice setup di render aggiungendo i COMP “Camera”, “Light”, “Geometry” e un TOP “Render”. Per questo esercizio possono tutti essere lasciati ai loro valori di default. Assicuratevi di aggiungere un SOP “In” al COMP “Geometry”, così che possiamo trasmettergli il nostro insieme di punti, e abilitate i segnalini di render e di visualizzazione nel SOP “In” dentro al COMP “Geometry”. Questo completa il passo 5.



Particle 2

Adesso creiamo un Operatore MAT “GLSL” e due DAT “Text”. Nel parametro “Operator” del DAT “Info” facciamo riferimento al MAT “GLSL”, per essere più comodi nel rintracciare eventuali errori. Chiamiamo “vertex” uno dei due DAT “Text”, e rinominiamo l’altro “pixel”: questi saranno gli shader Vertex e Pixel in GLSL. Facciamo riferimento a “vertex” nel parametro “Vertex Shader” del

MAT “GLSL”, e a “pixel” nel parametro “Pixel Shader”. Dobbiamo quindi fare riferimento al TOP “GLSL” che abbiamo creato. Per fare ciò, nella pagina “Samplers 1” dei parametri del MAT “GLSL” aggiungiamo “sPointPosition” al primo parametro “Sampler Name”, e aggiungiamo il nome della texture del TOP “Noise” nella casella “TOP”. Nel file di esempio è stato aggiunto un TOP “Null” chiamato “null1” dopo il TOP “GLSL”, ed è questo il nome dell’Operatore a cui si fa riferimento nel parametro “TOP”. Siate molto attenti con il parametro “Sample Name”, poiché questo sarà il nome da utilizzare nel codice, e se sarà differente non vedrete alcuna uscita, poiché non farete riferimento in modo corretto alla posizione delle particelle. Infine, nella pagina “Vectors 1” del MAT “GLSL”, aggiungiamo “uPointsPerInstance” nel primo “Uniform Name”, e inseriamo “1.0 / 1000” come primo valore del parametro “value0x”. Quest’ultimo vettore sarà utilizzato nello shader quando si campionano le texture dei punti nelle loro posizioni, per scalare gli indici dei punti da 0-1000 alle coordinate UV normalizzate, che vanno da 0.0 a 1.0. Con questo setup possiamo passare dal punto 6 al punto 7.

Da qui in avanti finiremo tutti i passaggi rimanenti all’interno degli shader GLSL. Prima di tutto modifichiamo “pixel”, il DAT “Text” che useremo come shader Pixel, e inseriamo il codice seguente:

```
layout(location = 0) out vec4 fragColor;

void main()
{
// shade pixel white
fragColor = vec4(1.0, 1.0, 1.0, 1.0);
}
```

Questo è uno shader Pixel molto semplice, come abbiamo visto in precedenza in questo capitolo, e tutto ciò che fa è rendere bianchi i pixel in ingresso. Questo conclude il passo 7.

Modifichiamo ora “vertex”, il DAT “Text” che useremo come shader Vertex, e inseriamo il codice:

```
// setup inputs
uniform sampler2D sPointPosition;
uniform float uPointsPerInstance;
in float pointIndex;

void main()
{
    // create the uv from point index
    vec2 uv;
    uv.x = (pointIndex * uPointsPerInstance) + (uPointsPerInstance * 0.5);
    uv.y = 0.5;

    // sample the noise texture using the uv
    vec4 newPosition = texture(sPointPosition, uv);

    // set point size to your liking
    gl_PointSize = 1.0;

    // move point from object space to screen space and output to gl_Position
    vec4 worldSpaceVert = TDDeform(newPosition);
    gl_Position = TDWorldToProj(worldSpaceVert);
}
```

Una volta salvato ed eseguito il codice vedrete il sistema di particelle creare l’effetto di un nastro utilizzando i valori casuali che abbiamo generato. Analizziamo questo shader Vertex.

La prime quattro righe definiscono la texture come uno “uniform” chiamato “sampler2D”, il valore “uPointsPerInstance come “uniform float” e l’attributo degli indici dei punti in ingresso come un valore decimale:

```
// setup inputs
uniform sampler2D sPointPosition;
uniform float uPointsPerInstance;
in float pointIndex;
```

Le linee di codice successive creano la mappa UV da usare quando si campiona la texture. Per creare la X delle UV prendiamo l’indice del punto in ingresso e lo moltiplichiamo per “uPointsPerInstance”, che è 1/1000. Questo ci restituisce il luogo da cui campionare in un range tra 0.0 e 1.0. Un fattore chiave da ricordare quando si crea una UV manualmente è che le coordinate UV hanno precisione infinita, quindi una UV di 0 lungo l’asse X non è il primo pixel, ma il bordo sinistro del primo pixel, causando degli errori nella visualizzazione, poiché lo shader cercherà di interpolare il 50% del primo

pixel e il 50% di qualsiasi cosa ci sia alla sua sinistra (che dipende dalle impostazioni del parametro "repeat"). Perciò abbiamo bisogno di spostare il nostro campione di metà di un "uPointsPerInstance", ed è il motivo per cui aggiungiamo il risultato di "uPointsPerInstance" moltiplicato per 0.5 al luogo che abbiamo calcolato moltiplicando "pointIndex" e "uPointsPerInstance".

Ricapitolando: 1. Abbiamo bisogno di convertire l'indice dei punti da 0-1000 alle coordinate UV che vanno da 0.0 a 1.0;

1. Facciamo ciò moltiplicando l'indice dei punti per il risultato di 1/1000, che ci restituisce il nostro intervallo nel range 0.0-1.0;
2. Aggiungiamo quindi metà del valore di "uPointsPerInstance" (che è metà di un singolo intervallo) per spostare il nostro campionamento, così che campioniamo il centro di ogni pixel, e non il bordo sinistro.

Infine, poiché sappiamo che la texture è alta solo un pixel, possiamo impostare "uv.y" a 0.5 (ancora una volta perché non vogliamo campionare il bordo sinistro dei pixel, ma il loro centro):

```
// create the uv from point index
vec2 uv;
uv.x = (pointIndex * uPointsPerInstance) + (uPointsPerInstance * 0.5);
uv.y = 0.5;
```

La prossima cosa da fare è utilizzare le coordinate UV per campionare la texture:

```
// sample the noise texture using the uv
vec4 newPosition = texture(sPointPosition, uv);
```

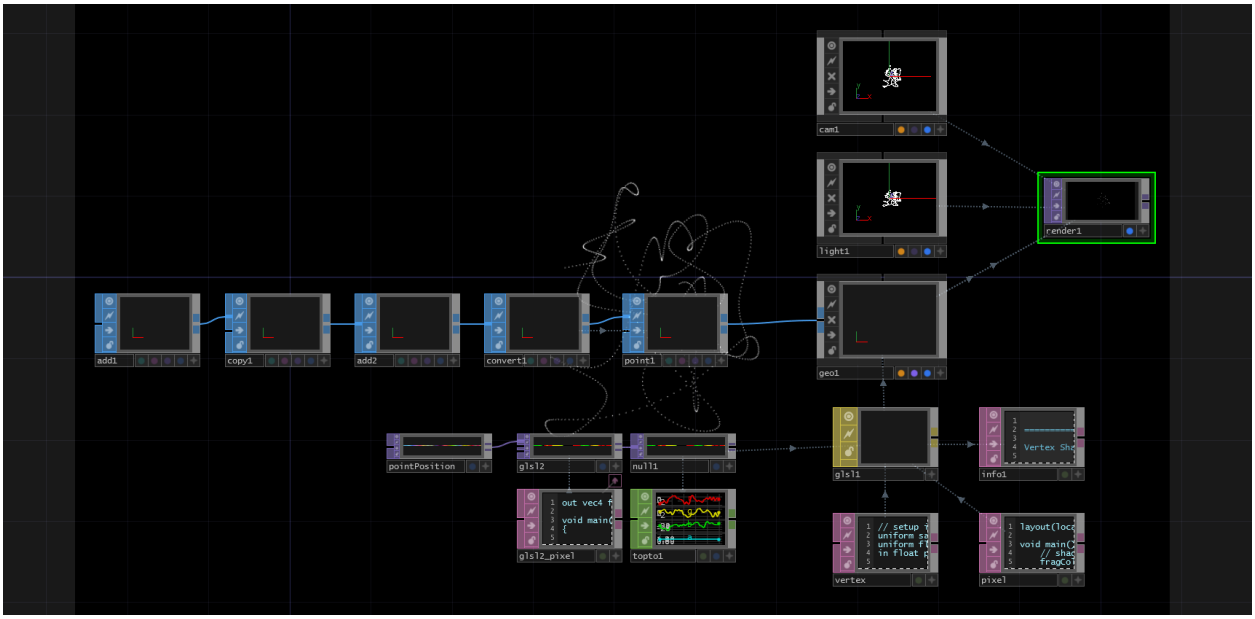
Prima cdi assegnare le nuove posizioni dei punti usiamo questo utile frammento di codice GLSL per regolare velocemente la dimensione delle particelle. Siamo in grado di farlo perché in precedenza abbiamo usato un SOP "Convert" per impostare la tipologia delle particelle su point sprite (dato che questo codice funziona solo con gli sprite).

```
// set point size to your liking
gl_PointSize = 1.0;
```

Infine, il codice di seguito prende i nostri valori "newPosition" dallo spazio dell'oggetto e usa "TDDeform()" per spostarlo nello spazio globale, quindi moltiplica la posizione di "uTDMat.cam" per spostare il punto nello spazio della telecamera, e infine "TDCamToProj()" è utilizzato per convertire lo spazio della telecamera in punti dello spazio dello schermo, assegnati a "gl_Position", che rappresenta l'uscita di default per ogni posizione dei punti.

```
// move point from object space to screen space and output to gl_Position
vec4 worldSpaceVert = TDDeform(newPosition);
gl_Position = TDWorldToProj(camSpaceVert);
```

Con questo abbiamo completato il primo obiettivo, muovere delle particelle con texture. Anche se questo può non sembrare un sistema di particelle tradizionale, i passaggi utilizzati costituiscono le fondamenta per le prossime implementazioni.



Particles final

12.7.3 Utilizzare Geometrie come Sorgenti

Ora che abbiamo una comprensione basilare su come muovere delle particelle con texture, possiamo aggiungere una geometria e sfruttare i suoi punti come posizioni iniziali delle particelle. In questo esercizio sostituiremo il nostro SOP “Add” con un SOP “Grid” (con lo stesso numero di punti) e aggiungeremo dei valori casuali alla posizione di ogni particella. Ci sono alcuni passaggi che dovremo seguire:

1. Creare una texture dai dati delle posizioni dei punti del SOP “Grid”;
2. Usare questa texture per collocare le nostre particelle nelle stesse posizioni dei punti del SOP “Grid”;
3. Applicare la texture appena creata sulle nuove posizioni per modificare la griglia.

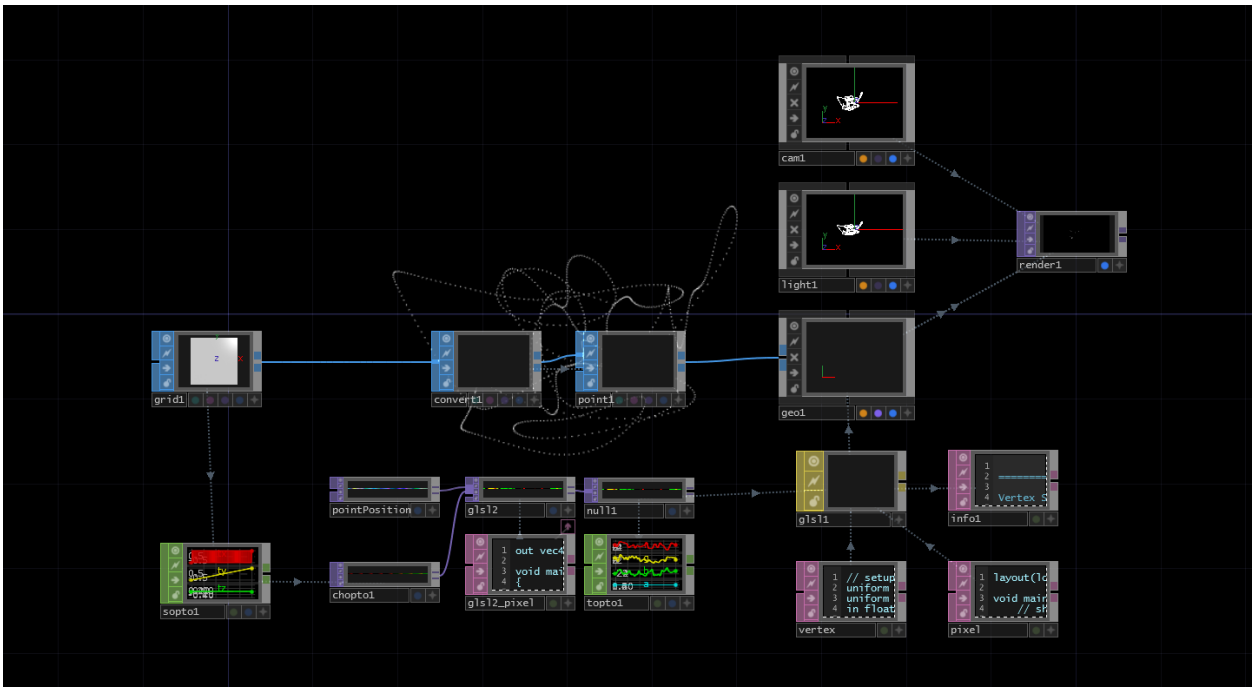
È consigliato procedere nella lettura mentre si osserva l’esempio “02_Source_geometry.toe”, poiché faremo riferimento ad alcuni Operatori presenti nel progetto chiamandoli per nome.

Il primo passo è creare una texture dai dati delle posizioni dei punti del SOP “Grid”. La sorgente delle posizioni dei punti nel primo esempio era un SOP “Add”, un SOP “Copy” e un altro SOP “Add”: iniziamo rimuovendo questi Operatori e sostituendoli con un SOP “Grid”, con il parametro “Rows” impostato a 100 e il parametro “Columns” impostato a 10. Questa combinazione di righe e colonne creerà una griglia con lo stesso numero di punti del nostro esempio precedente.

Il passo successivo è prendere tutte le posizioni dei punti dal SOP “Grid” utilizzando un “SOP to CHOP”: creiamo un “SOP to CHOP” impostando il parametro “SOP” con il nome del SOP “Grid”, che in questo caso è “grid1”.

Così avremo un CHOP con tutte le posizioni dei punti come canali separati. Possiamo trasformare questi canali XYZ in canali RGB di una texture utilizzando un Operatore “CHOP to TOP”: creiamo un “CHOP to TOP” e impostiamo il parametro “CHOP” con il nome del “SOP to CHOP”, che in questo esempio dovrebbe essere “sopto1”. Assicuratevi di impostare il “Pixel Format” in “32-bit float (RGBA)” nelle impostazioni “Common” del “CHOP to TOP”, visto che trasmetteremo questo tipo di dati allo shader GLSL, e vogliamo che continui a produrre una texture a 32-bit. Colleghiamo l’uscita del “CHOP to TOP” al secondo ingresso di “glsl2”, il TOP “GLSL” che nell’esempio precedente stavamo usando per scalare i valori della texture.

Questo completa il primo passaggio dell’esempio:



Ora che abbiamo una texture, facciamo alcune aggiunte al nostro shader. Di seguito riportiamo lo shader finale dell’esempio precedente:

```
out vec4 fragColor;

void main()
{
// sample the input
vec4 inPosition = texture(sTD2DInputs[0], vUV.st);

// scale each color channel (the XYZ of the vector) separately
vec4 outPosition = vec4(inPosition.x * 5.0 - 2.5, inPosition.y * 5.0 - 2.5, inPositi\
on.z * -5.0 - 20.0, 1.0);

// output the new position
fragColor = outPosition;
}
```

Inizieremo aggiungendo una linea per campionare la nuova texture con i dati delle posizioni del SOP "Grid". Dopo la linea 7 inserite questo (ricontrolleremo il codice completo alla fine):

```
vec4 gridPosition = texture(sTD2DInputs[1], vUV.st);
```

Questa aggiunta crea un vettore a 4 componenti con i nostri dati XYZ collegati al secondo ingresso (ricordate che gli ingressi sono enumerati a partire da 0). Se volete visualizzarlo in maniera rapida, cambiate l'ultima riga temporaneamente:

```
fragColor = gridPosition;
```

Questo sposterà tutte le particelle sui punti statici del SOP "Grid". Prima di continuare, assicuratevi di rimettere l'ultima riga come:

```
fragColor = outPosition;
```

Ora ci concentreremo su questa linea del codice:

```
vec4 outPosition = vec4(inPosition.x * 5.0 - 2.5, inPosition.y * 5.0 - 2.5, inPositi\
on.z * -5.0 - 20.0, 1.0);
```

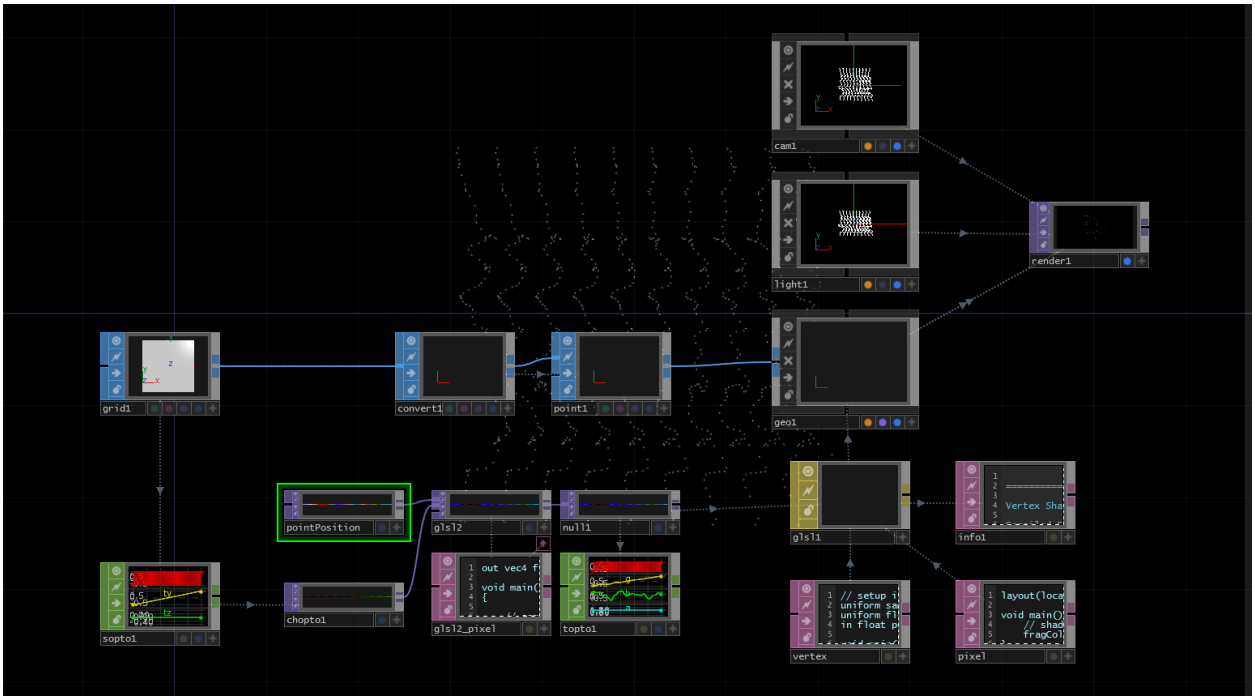
In precedenza prendevamo i valori del TOP "Noise", li scalavamo per renderli più interessanti, e quindi li spostavamo per farli rientrare nell'inquadratura della telecamera. Ora il nostro obiettivo è prendere le posizioni della griglia ed effettarle con gli stessi valori. Per fare ciò, possiamo usare una riga come questa:

```
vec4 outPosition = vec4(gridPosition.x + (inPosition.x * 0.1), gridPosition.y + (inP\
osition.y * 0.1), gridPosition.z + inPosition.z, 1.0);
```

All'interno del "vec4" prendiamo i valori XYZ del SOP "Grid" e li aggiungiamo agli XYZ della texture "Noise". L'unica cosa che qui abbiamo fatto in più, è che prima di sommare i valori X e Y del "Noise" li scaliamo, riducendoli in modo da rendere più agevole vedere la forma del SOP "Grid" nel render. Il codice dello shader completo dovrebbe essere così:

```
out vec4 fragColor;

void main()
{
// sample the inputs
vec4 inPosition = texture(sTD2DInputs[0], vUV.st);
vec4 gridPosition = texture(sTD2DInputs[1], vUV.st);

// add scaled noise texture values to the grid position values
vec4 outPosition = vec4(gridPosition.x + (inPosition.x * 0.1), gridPosition.y + (inP\
osition.y * 0.1), gridPosition.z + inPosition.z, 1.0);

// output the new position
fragColor = outPosition;
}
```

Una volta che avete salvato, dovreste vedere la griglia modificata dalla texture del TOP "Noise":



Sentitevi liberi di sperimentare sostituendo il SOP “Grid” con altre geometrie con 1000 punti.

12.7.4 Aggiungere la Velocità

In questa sezione elimineremo i valori casuali che muovono le particelle e aggiungeremo una velocità costante. Non investiremo molto tempo ad approfondire i concetti fisici, perciò se per voi sono nuovi vi raccomandiamo le seguenti risorse:

- Nature of Code[14] un ottimo libro di Processing riguardo la simulazione delle forze naturali
- Khan Academy[15] per imparare i concetti fisici

Creeremo anche una semplice interfaccia utente che ci permetterà di aggiungere una velocità costante alle particelle nello spazio XYZ. È importante sapere come sono controllate le particelle, perché i dati dell’ultimo fotogramma dovranno essere aggiunti al frame corrente per ricavare le nuove posizioni. Pensiamo ad una particella che si sposta per alcuni frame: per ogni fotogramma deve sapere dove si trovava nel frame precedente, così da poter aggiungere la velocità e calcolare la sua nuova posizione; nell’esempio precedente i valori casuali coinvolgevano solamente il fotogramma corrente.

L’elemento principale che dobbiamo aggiungere è un loop di feedback così che possiamo fornire continuamente i dati dell’ultimo frame, aggiornare la texture con le nuove posizioni e portare nuovamente i dati nell’input.

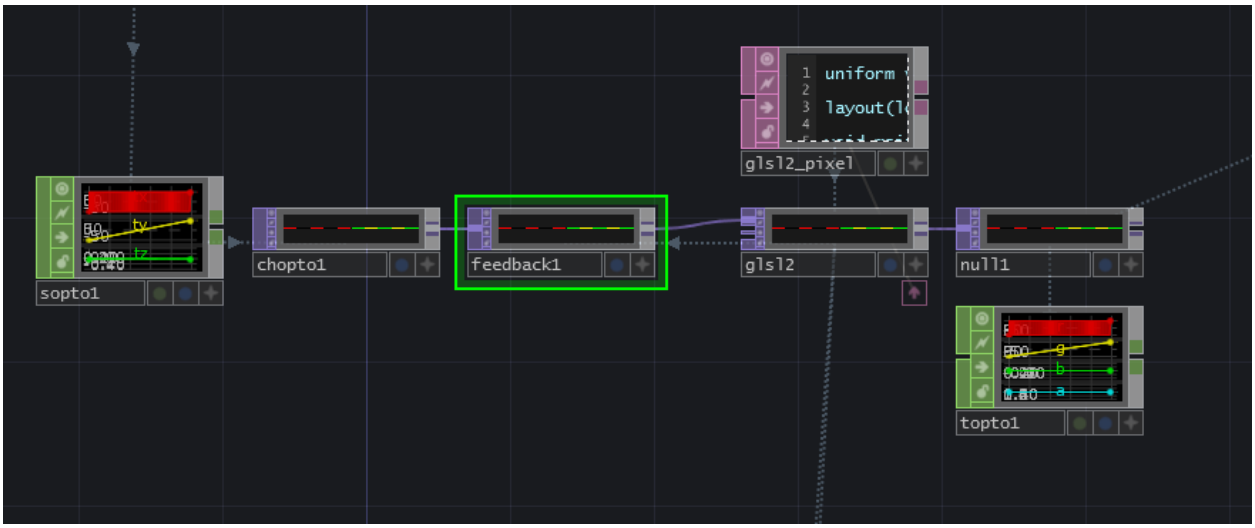
[14] http://natureofcode.com/
[15] https://www.khanacademy.org/science/physics

Potete seguire l'esempio `01_adding_velocity.toe` nella cartella `TouchDesigner Example Files/12.7.4`.

Iniziamo cancellando il TOP "Noise" e scollegando il "chopto1" dal TOP "GLSL". Seguiamo questi passaggi:

1. Creiamo un TOP "Feedback";
2. Colleghiamo l'uscita di "chopto1" all'ingresso del TOP "Feedback";
3. Colleghiamo l'uscita del TOP "Feedback" al primo ingresso del TOP "GLSL";
4. Impostiamo il parametro "Target TOP" del TOP "Feedback" con il nome del TOP "GLSL" (nel file di esempio si chiama "glsl2").

Dovrebbe apparire come l'immagine seguente:



Ora creiamo una nuova Uniform nella pagina "Vectors 1" dei parametri del TOP "GLSL". Chiamiamola "uVel" e lasciamo i valori a "0".

Saltando più avanti potete vedere lo shader finito, ma qui spiegheremo i singoli cambiamenti da apportare.

Aggiungiamo una riga all'inizio dello shader per dichiarare la nuova Uniform:

```
1  uniform vec3 uVel;
```

Cambieremo il nome del nostro vec4 in uscita da "fragColor" in "oPosition", un'abbreviazione per "output position".

Quindi, invece di campionare le posizioni casuali e della griglia, campioneremo la nuove posizioni in ingesso, portate nello shader dal TOP "Feedback".

```
vec4 pos = texture(sTD2DInputs[0], vUV.st);
```

Aggiungiamo il nostro nuovo valore della velocità alla posizione precedente di ogni punto:

```
pos.xyz += uVel.xyz;
```

E infine creiamo un output con le nuove posizioni:

```
oPosition = pos;
```

Lo shader finito per questo esempio dovrebbe apparire così:

```
uniform vec3 uVel;

out vec4 oPosition;

void main()
{
    // get input positions
    vec4 pos = texture(sTD2DInputs[0], vUV.st);

    // add our single velocity values to every point position
    pos.xyz += uVel.xyz;

    // output the new point position
    oPosition = pos;

}
```

Gli ultimi elementi di cui abbiamo bisogno sono un'interfaccia per cambiare la Uniform "uVel", e un pulsante per reimpostare le particelle resettando il feedback.

Nell'esempio abbiamo creato uno slider 2D per le velocità XY delle particelle e uno slider unidimensionale per la velocità lungo Z. Potete sperimentare con altri tipi di slider e pulsanti, purché facciate riferimento ai valori dei canali nei primi tre campi della Uniform "uVel" sulla pagina "Vectors 1" dei parametri del TOP "GLSL".

Lo script da aggiungere al pulsante di reset cambierà in base al tipo di interfaccia che creerete, ma alla fine deve sempre esserci la stessa riga. Questa linea attiverà il parametro "Reset" del TOP "Feedback", che quindi reimposterà il feedback restituendo la posizione iniziale dei punti sulla griglia. Nello script di esempio tutti gli elementi dell'interfaccia vengono resettati nella posizione "0" e quindi anche il TOP "Feedback" è resettato.

12.7.5 Velocità Casuali

Ora abbiamo il sistema di particelle più semplice che possiamo immaginare: un sistema dove tutte le particelle si muovo alla stessa velocità costante. In questa sezione daremo ad ognuno dei punti una propria velocità casuale invece di controllarli con degli elementi di un'interfaccia. Ciò creerà l'effetto di un'esplosione di particelle dal SOP "Grid".

Iniziamo rimuovendo la Uniform "uVel" dal TOP "GLSL" ed eliminando i parametri che fanno riferimento all'interfaccia utente. La pagina "Vectors 1" dei parametri del TOP "GLSL" dovrebbe essere vuota.

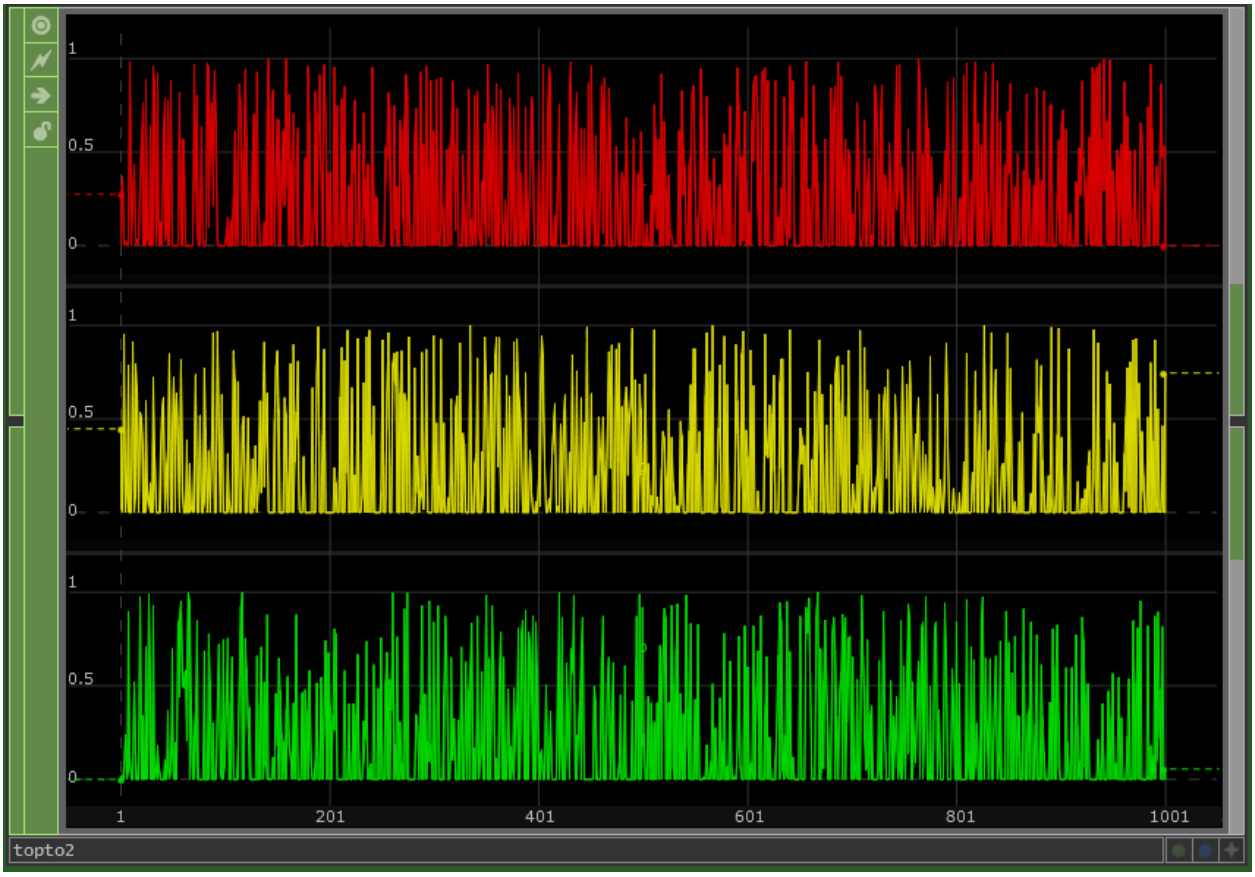
Quindi cancelliamo gli slider e gli elementi dell'interfaccia utente creati in precedenza per controllare la velocità delle particelle, *ma non eliminiamo il pulsante di reset*, poiché continueremo ad usarlo all'interno di questo esempio.

A seconda di quali elementi abbiamo creato per controllare le particelle, dovremo rimuovere anche il codice Python associato ad essi dal DAT "Panel Execute" collegato al pulsante di reset. All'interno della funzione "def offToOn" dovremmo avere solo una riga che resetta il TOP "Feedback":

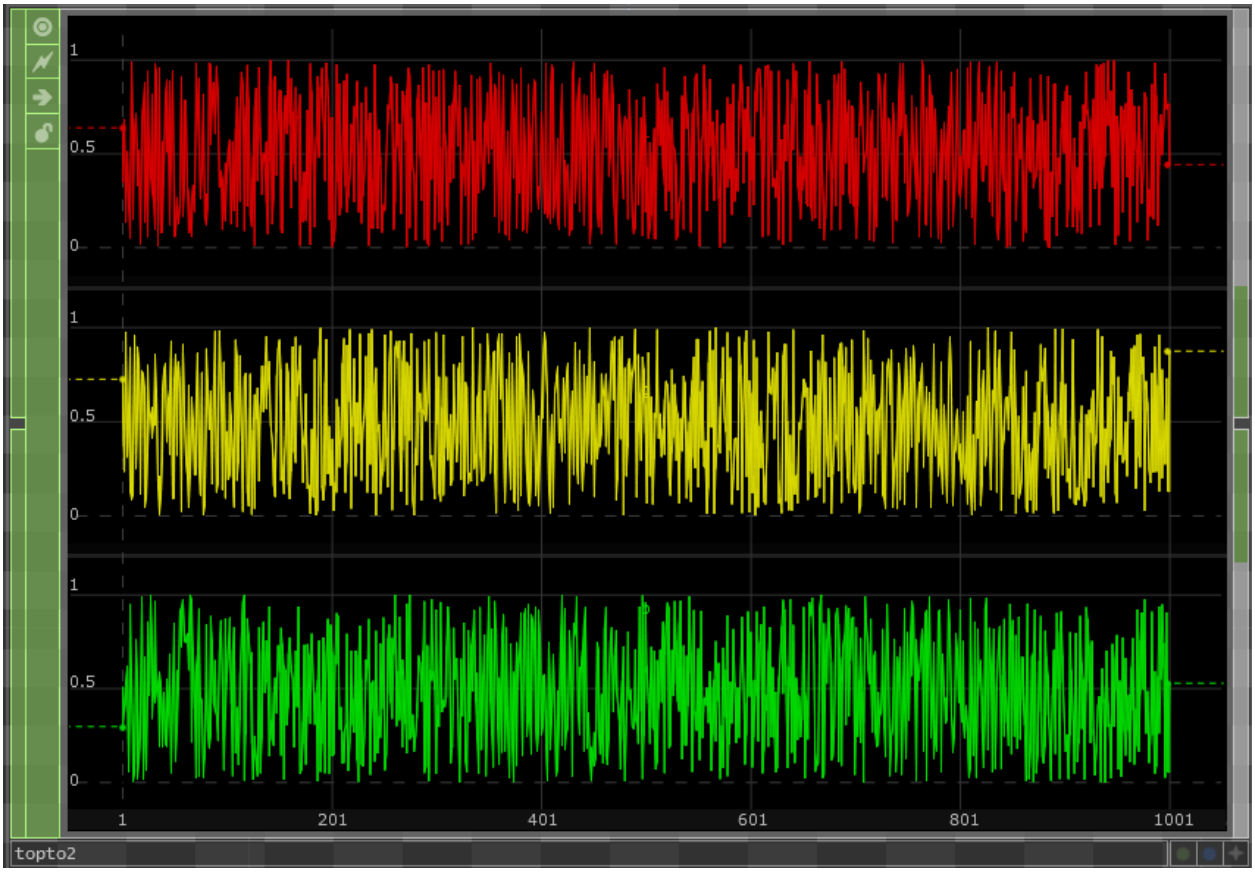
```
1 op('feedback1').par.resetpulse.pulse()
```

L'ultimo elemento di cui abbiamo bisogno è un TOP "Noise" con una risoluzione di 1000x1 pixel, uguale alla risoluzione del nostro TOP "CHOP to". Impostiamo la tipologia del TOP "Noise" in "Random (GPU), attiviamo il tasto "Monochrome" e impostiamo i parametri "Amplitude" a 0.5 e "Offset" a 0.5. Cambiare questi due parametri è un modo facile per spostare i valori casuali da un range di "0-1" con un offset nullo, ad un range di "0-1" centrato in 0.5.

Visivamente, ci stiamo spostando da questo range:



A questo:



Colleghiamo ora il TOP “Noise” al secondo ingresso del TOP “GLSL”.

Il network dovrebbe apparire così:

Nel nostro shader dobbiamo apportare solo pochi cambiamenti.

Dopo la riga in cui campioniamo le posizioni in ingresso dalla griglia, aggiungeremo una linea che campiona la nostra texture casuale e crea un vec4 chiamato “velocity”:

```
vec4 velocity = texture(sTD2DInputs[1], vUV.st) * 2 - 1;
```

Questo procedimento dovrebbe ormai apparirci famigliare. I valori “*2-1” alla fine della riga sono delle semplici operazioni matematiche per cambiare il range da “0-1” ad un range che parta da “-1” e arrivi a “1”.

Nella prossima linea di codice, invece di aggiungere la Uniform “uVel” dobbiamo aggiungere il vettore “velocity”:

```
pos.xyz += velocity.xyz;
```

Ora possiamo fare click sul tasto di reset e osservare il sistema di particelle esplodere allontanandosi dai punti del SOP “Grid”. Sperimentate con i valori del TOP “Noise” e la matematica che regola il range nello shader per osservare come sia possibile ottenere risultati differenti.

12.8 Come Proseguire?

Questo capito dovrebbe aver aiutato a districare un po' dei misteri iniziali del GLSL, pur senza diventare eccessivamente tecnico.

Se volete continuare ad imparare GLSL per sperimentare con scene 3D e shader dei materiali più complessi, o con strumenti di compositing 2D più avanzati e texture generative, di seguito riportiamo alcune risorse:

1. Pagina di riferimento principale di OpenGL: http://www.opengl.org/wiki/Main_Page[16]
2. Informazioni generali su OpenGL: http://www.opengl.org/wiki/General_OpenGL[17]
3. Wiki di Derivative: http://www.derivative.ca/wiki088/index.php?title=Glsl[18]
4. Shader Toy: https://www.shadertoy.com/[19]

[16] http://www.opengl.org/wiki/Main_Page
[17] http://www.opengl.org/wiki/General_OpenGL
[18] http://www.derivative.ca/wiki088/index.php?title=Glsl
[19] https://www.shadertoy.com/

13 Progetti

Questa sezione verte sulle risorse disponibili al di fuori del libro stesso. Nella cartella “Projects” sono presenti due sottocartelle, “Beginner” e “Advanced”. Ognuna contiene una serie di tutorial che includono file dei progetti e video tutorial.

Alcuni di questi esempi saranno di natura pi tecnica, mentre altri si riveleranno pi artistici, ma tutti quanti cercheranno di evidenziare un concetto o una funzione comune.

I video partiranno da un progetto vuoto, e lo comporranno in maniera completa, evidenziando di volta in volta l’approccio adottato per completare i diversi compiti.

Crediti

In ordine di contributo:

Elburz Sorkhabi - autore e creatore originario, editore

Byron Wong - management e operazioni

Matthew Hedlin - autore del capitolo 12.6 - Importare Shadertoy, correttore di bozze e editing, conversione al formato GitBook

Greg Hermanovic - introduzione

Malcolm Bechard - correttore di bozze, editore

Ben Voigt - correttore di bozze, editore

Davide Santini - traduttore all'italiano