ALMA MATER STUDIORUM - UNIVERSITÀ DI BOLOGNA

SCUOLA DI INGEGNERIA E ARCHITETTURA

DIPARTIMENTO DI INFORMATICA – SCIENZA E INGEGNERIA

CORSO DI LAUREA TRIENNALE IN INGEGNERIA INFORMATICA

TESI DI LAUREA

In

CALCOLATORI ELETTRONICI - T

SVILUPPO IN UNITY 3D DI UN BENCHMARK PER LA STIMA DEL FLUSSO OTTICO

CANDIDATO:

Simone Dosi

RELATORE:

Prof. Stefano Mattoccia

CO-RELATORI:

Dott. Fabio Tosi
Dott. Matteo Poggi
Dott. Filippo Aleotti

Anno Accademico 2018/19

Sessione II

Indice

Introduzione

Negli ultimi anni, i motori grafici per lo sviluppo di videogiochi (Unity e Unreal Engine 4 principalmente) stanno suscitando sempre più interesse da parte della comunità di sviluppatori specializzati nel machine learning e nella computer vision.
Questo crescente interesse è dovuto alla possibilità, offerta da questi motori grafici, di generare scene utili a testare e\o addestrare in situazioni controllate i propri algoritmi.
Un esempio di framework basato sui moderni motori grafici è CARLA (carla.org). Esso fornisce gli strumenti per sviluppare, addestrare e validare algoritmi di guida autonoma ed è basato sul motore grafico Unreal Engine 4.

Lo scopo di questa tesi è quello di creare un framework tramite il quale generare video sintetici di scene fluviali. Questi serviranno a misurare l'accuratezza di algoritmi di Optical Tracking Velocimetry (OTV) per il tracciamento di corpi sulla superficie di un corso d'acqua [1].
Le scene fluviali dovranno essere parametrizzabili per rendere possibile la generazione di video dalle caratteristiche diverse e testare gli algoritmi in varie condizioni.
I parametri della scena, oltre a poterne variare l'aspetto, fungono da valori di confronto con i risultati degli algoritmi di OTV. Essi sono infatti considerati valori di "ground-truth" e insieme ai dati forniti dal framework sui singoli traccianti ci permettono un'accurata valutazione degli algoritmi di OTV.
Questa tesi si radica sul lavoro da me svolto durante il mio tirocinio universitario [2] e ne continua l'attività andando a sviluppare un nuovo framework più' completo e basato totalmente su assets di pubblico dominio.

Il nuovo framework è sviluppato come applicativo Windows, MacOS e Linux ed è basato sul motore grafico Unity. Al framework ho dato il nome di "River flow speed benchmark" (RFSB) e di seguito è riportato il suo logo da me disegnato.

RFSB servirà a renderizzare una scena fluviale in computer grafica tridimensionale sulla quale si muovono schiume, foglie e altri corpuscoli che fungeranno da traccianti per gli algoritmi di OTV.
I video sintetici non vengono esportati direttamente; essi devono essere generati tramite applicativi di terze parti capaci di effettuare una registrazione dello schermo.

La tesi sarà strutturata in modo da analizzare gli strumenti utilizzati per la creazione dell'applicativo del framework. Successivamente analizzeremo le scelte implementative, cercando di approfondire i punti salienti utili ad eventuali sviluppi futuri. In conclusione, utilizzando i video esportati dall'applicativo, sarà valutata insieme al laureando Riccardo Turra la capacità degli algoritmi di OTV di fornire misurazioni accettabili considerando i valori di "ground-truth" forniti dal framework sviluppato.

Sono riportati 2 frame tratti da il video 6 del test effettuato dal laureando Riccardo Turra. Entrambi i frame sono estrapolati al secondo 22, il primo è tratto dal video originale e il secondo è frutto delle operazioni di tracciamento dell'algoritmo OTV.

Lo scopo secondario di questa tesi è quello di fungere da documentazione per futuri utilizzatori e sviluppatori del framework. Saranno riportati man mano i casi d'uso principali, dando per scontata una conoscenza basilare degli strumenti e dei linguaggi utilizzati nello sviluppo.

Strumenti utilizzati

2.1 Unity

2.1.1 Cos'è Unity?

Unity è un motore grafico 3D pensato per lo sviluppo di videogiochi che include strumenti avanzati (es. editor di terreni, editor di interfacce, gestore di pacchetti etc.) e risulta attualmente uno dei motori grafici più usati al mondo, attirando a sé sempre più interessi da parte dei settori dell'automotive, cinematografia digitale e architettura.
Internamente a Unity è possibile programmare la logica degli applicativi in linguaggio C# e gli shader in linguaggio HLSL/CG oppure tramite il linguaggio visuale a nodi "shader-graph".

2.1.2 Perchè Unity?

Unity è stato scelto come motore grafico per lo sviluppo del framework per vari motivi:

- La personale pregressa familiarità con il motore grafico e con il linguaggio C#
- Il linguaggio C# risulta più facile da imparare e trattare da un nuovo sviluppatore rispetto al C++, uno dei linguaggi più usati nei motori grafici concorrenti (ad esempio Unreal Engine 4)
- Il motore grafico e rispettivi strumenti sono ben documentati
- E' presente una quantità praticamente illimitata di materiale didattico sia sul motore Unity che sul linguaggio C#
- Possiede uno store di assets, denominato "Unity Asset Store", da cui è possibile acquistare componenti, scripts, modelli 3D etc.
  All'interno di questo negozio virtuale è possibile trovare contenuti gratuiti pubblicati direttamente da Unity.
- Offre la possibilità di generare eseguibili nativi verso un numero molto esteso di piattaforme. In questo caso era di massima importanza avere supporto nativo alle piattaforme PC (Windows, Mac OS X e Linux).

2.1.3 Installare Unity

Unity è disponibile sotto forma di applicativo per le piattaforme Windows e Mac OS X. Il download e i requisiti di sistema del motore grafico sono disponibili sul sito unity3d.com/get-unity/download.
Unity è disponibile in varie licenze ognuna con prezzo e modalità d'uso differenti. La licenza di nostro interesse è quella denominata "Personal", completamente gratuita per tutte le realtà non a scopo di lucro, richiede solamente la registrazione di un account utente a cui assegnare la licenza.
Il motore grafico è disponibile in varie versioni. Per facilitarne la gestione, Unity mette a disposizione il programma "Unity Hub" che, una volta installato, permette di scaricare e gestire le versioni del motore grafico di interesse.

2.1.4 Progetto dell'applicativo

Il progetto del framework è aggiornato all'ultima versione beta di Unity al momento della scrittura di questa tesi, ovvero la 2019.3 beta 12.
L'utilizzo delle versioni instabili (beta/alpha) è generalmente sconsigliato, tuttavia ho proceduto in questo modo per ottenere l'accesso ad alcuni miglioramenti presenti in questa versione e garantire il corretto funzionamento con la ormai imminente versione 2019.3 stabile. Consiglio ai futuri sviluppatori di utilizzare una versione stabile del motore grafico per evitare di imbattersi in eventuali problematiche.
Il progetto è basato sulla Universal Render Pipeline (URP), la nuova pipeline di rendering che Unity sta sviluppando da qualche anno a questa parte per andare a sostituire quella attuale. La scelta di basare il progetto su questa nuova tecnologia è maturata per garantire un compatibilità nativa con le future versioni del motore grafico e sfruttare nuovi strumenti integrati come lo shader-graph.
Lo shader-graph, in particolare, è lo strumento con cui è stato scritto lo shader del fiume del framework. Esso è un editor visuale che permette di creare shader in modo intuitivo tramite un'interfaccia a nodi con anteprime in real-time.
Maggiori informazioni sulla Universal Render Pipeline e sullo shader-graph sono disponibili sul documentazione ufficiale di Unity (docs.unity3d.com).

2.2 Github e Git

2.2.1 Cos'è Github?

Github è un servizio di hosting per progetti software, offerto dall'omonima azienda, che permette di gestire attraverso il software di controllo di versione Git i propri progetti.
Esso risultava una buona soluzione per gestire il progetto dell'applicativo in quanto permetteva di possedere gratuitamente una repository remota, oltre che gestire eventuali membri di un team di sviluppo e i loro relativi permessi di accesso alla repository.

2.2.2 Come operare sulla repository?

Per lavorare da remoto sulla repository è necessario un client Git e una conoscenza basilare del funzionamento di Git.
Si procede a esporre il flusso di lavoro consigliato per operare sulla repository:

- Il ramo *master* contiene i merge funzionanti e dichiarati stabili
- Il ramo *development* contiene il codice in fase di sviluppo.
  Su questo ramo convergono tutte le funzionalità una volta che sono terminate.
- Ogni nuova funzionalità è sviluppata in un ramo dedicato partendo dall'ultimo commit del ramo *development*.
- Gli sviluppatori sono tenuti a testare le proprie funzionalità prima di richiederne l'inclusione sul ramo *development*.
- Ogni commit deve possedere un titolo e una descrizione che spieghino in modo esaustivo le modifiche apportate.
- E' consigliabile che ogni commit effettuato contenga modifiche complete, in modo da facilitare la loro estrazione in caso di necessità.
  In caso contrario, è richiesto che il commit venga contrassegnato nel titolo con il prefisso "WIP" (work in progress).

Modellazione di scene realistiche per valutazione del flusso idrico con Unity

3.1 Scena

3.1.1 Generalità sull'implementazione

Il progetto dell'applicativo di benchmark è stato sviluppato attorno ad un'unica scena Unity ritrovabile all'interno del progetto al percorso:
Assets > Scenes> BenchmarkScene.unity

La scena è stata suddivisa al suo interno in varie macro-aree logiche:

- *Cameras*, area contenente gli oggetti che si occupano del rendering come camere (mono e stereo) e il volume di post-processing.
- *Lights*, area contenente *Sun Directional Light*, ovvero l'oggetto che si occupa di emettere la luce "solare" sulla scena.
- *Environment*, area contenente tutti gli oggetti che compongono il paesaggio.
  Questa sezione contiene due oggetti fondamentali per la resa grafica della scena, ovvero il terreno (oggetto *Terrain*) e il water system (oggetto *RFSB Water System*). Questi 2 oggetti verranno analizzati in seguito, ognuno nella sua sezione dedicata.
- *UI*, area contenente l'interfaccia grafica implementata attraverso la tecnologia *Canvas* di Unity.
- *Benchmark*, singolo oggetto dotato del componente *Application Controller*. Quest'ultimo è il centro nevralgico dell'applicativo e rappresenta il controller di tutti i suoi sottosistemi.

3.1.2 Terreno

Il terreno all'interno dell'applicativo è sviluppato utilizzando il componente *Terrain* nativo di Unity. Esso copre un'area di 40m x 40m.
La geometria del terreno è stata modellata attraverso l'uso dello script *RiverGeneratorFromPath.cs*.
Questo script richiede alcuni parametri per funzionare correttamente:

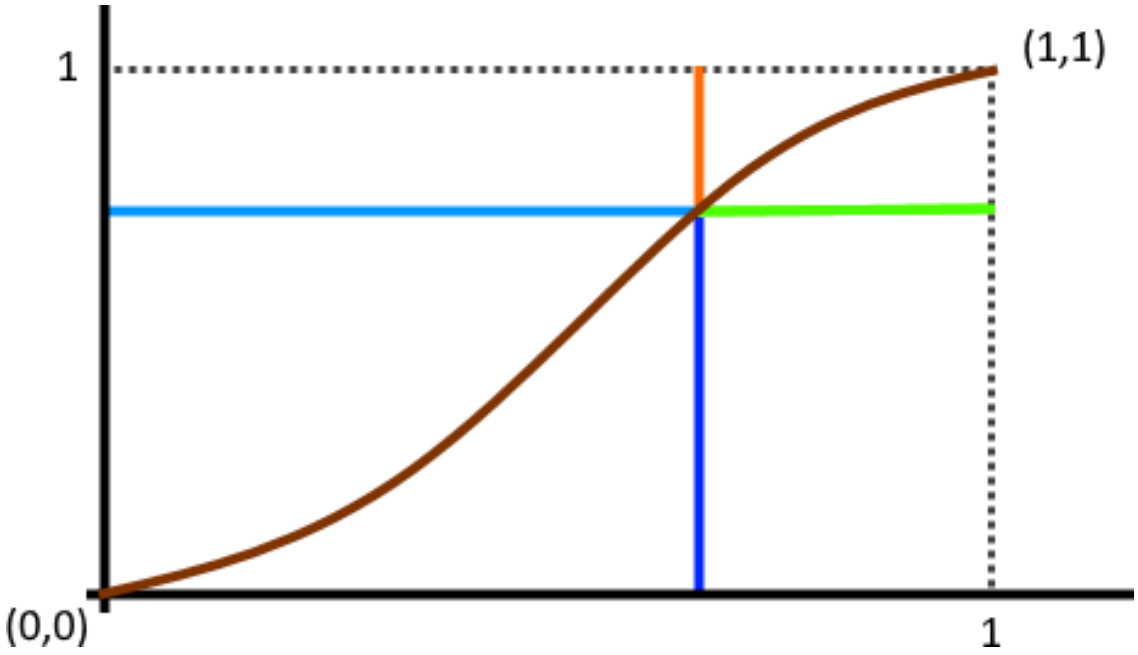
- *Path Creator*, riferimento ad un componente *Path Creator* presente in scena.
  Il componente *Path Creator* è parte di una libreria pubblica denominata *"Path*

Creator” [4]. Essa permette di disegnare all’interno della scena di Unity delle curve di Bézier [5] ed è stata utilizzata per definire il corso che il fiume avrebbe dovuto seguire.

Sopra è riportata la schermata dell’editor di scene di Unity in cui è visibile la curva di Bézier che determina il corso del fiume.

- *River Width*, valore numerico che definisce la larghezza in metri del corso del letto del fiume
- *River Border Width*, valore numerico che definisce la larghezza in metri delle spiagge che seguono il corso del fiume
- *Water Plane Height,* valore numerico che definisce l’altezza in metri tra il punto ad altezza massima del terreno e il piano dell’acqua
- *Water Max Depth*, valore numerico che definisce la profondità massima in metri del letto del fiume
- *Depth Curve*, curva che definisce la geometria in sezione del letto del fiume.
  La curva è rappresentata attraverso il tipo *AnimationCurve* nativo di Unity ed è editabile in modo intuitivo attraverso un editor fornito da Unity.
  Questa curva è utilizzata per calcolare l’altezza dei punti che compongono il letto del fiume.
  Per ogni punto si calcola la distanza tra esso e la curva di Bézier rappresentante il fiume. La distanza ottenuta viene espressa in un intervallo tra 0 e 1, dove 0 è il punto giacente sulla curva e 1 è un punto a distanza maggiore o uguale alla metà della larghezza del letto fiume sommata alla larghezza delle spiagge del fiume. Il valore ottenuto è il valore presente sulle ascisse del grafico della curva.

Ad ogni valore delle ascisse la curva fa corrispondere un valore sulle ordinate nell'intervallo tra 0 e 1 che rappresenta la distanza di quel punto dalla superficie del terreno. Il valore 0 sulle ordinate corrisponde ad un punto di massima profondità mentra il punto a valore 1 corrisponde ad un punto di altezza massima.
Di seguito riporto una rappresentazione grafica dello spaccato del fiume sul piano *XY* dove metto in mostra il significato geometrico dei parametri dello script descritti fin ora.



Il grafico rappresenta lo spaccato del fiume, in particolare, la curva marrone all'interno del sistema di cordinata è una possibile rappresentazione del parametro Depth Curve. I segmenti arancione, verde, blu e azzurro rappresentano in ordine i parametri Water Plane Height, River Border Width, Water Max Depth e mezza River Width.
I parametri rappresentati dai segmenti non sono nella scala della Depth Curve (0,1), essi sono inseriti nel grafico per mettere in evidenza i loro rapporti di misura.

- *Flowmap Export Path*, stringa che definisce il percorso dove lo script esporta la flow-map una volta completate le operazioni di modifica del terreno.
  La flow-map è una particolare classe di texture che mappa al suo interno un campo vettoriale bidimensionale rappresentante il flusso del fiume in ogni suo punto. Approfondiremo l'uso della flow-map quando sarà analizzato come essa viene utilizzata per creare l'effetto di movimento all'interno dello shader dell'acqua e definire il movimento dei traccianti nel corso del fiume.

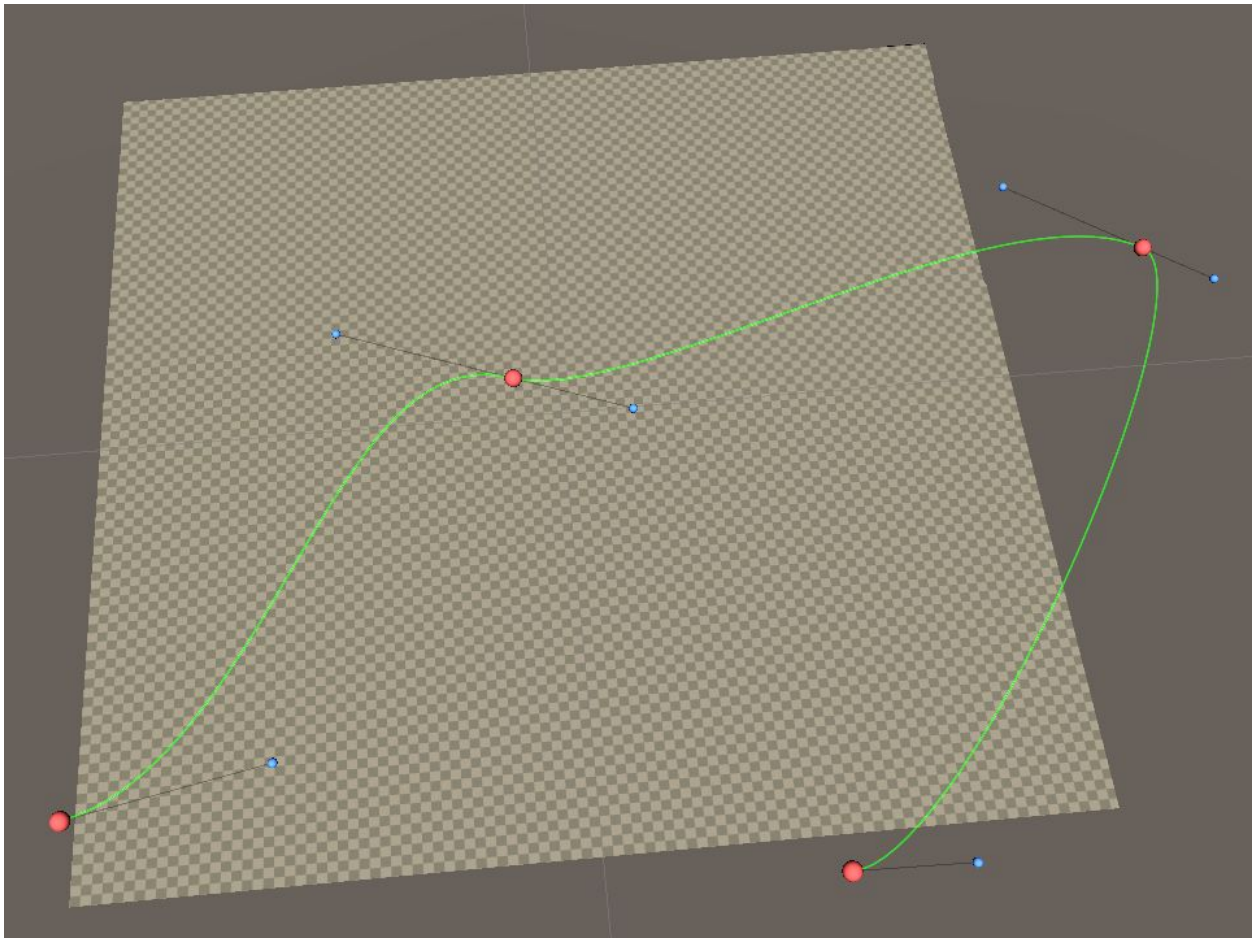
- *Heightmap Export Path*, stringa che definisce il percorso dove lo script esporta la height-map una volta completate le operazioni di modifica del terreno.
  La height-map è un particolare tipo di texture in scala di grigi utilizzata per mappare l'altezza di un terreno.
  In particolare, ad ogni punto del terreno di altezza massima viene associato un pixel di colore bianco mentre ad un punto di altezza minima un pixel di colore nero. A tutte le altezze intermedie viene associato un valore in scala di grigio.
- *Depthmap Export Path*, stringa che definisce il percorso dove lo script esporta la depth-map una volta completate le operazioni di modifica del terreno.
  La depth-map è una classe di texture avente mappatura dei dati opposta a quella della height-map. Avremo quindi i punti a minima altezza corrispondenti a pixel bianchi e quelli a massima altezza corrispondenti a pixel neri.

Una volta settati tutti i parametri è possibile lanciare l'esecuzione dello script tramite l'apposito bottone in editor.
L'esecuzione dello script si concretizza nella chiamata del suo metodo *GenerateRiver()* che si occuperà della modifica morfologica del componente Terrain e dell'esportazione delle texture di utilità per gli algoritmi che vedremo successivamente.
L'esecuzione del metodo *GenerateRiver()* si concretizza in pochi passaggi logici:

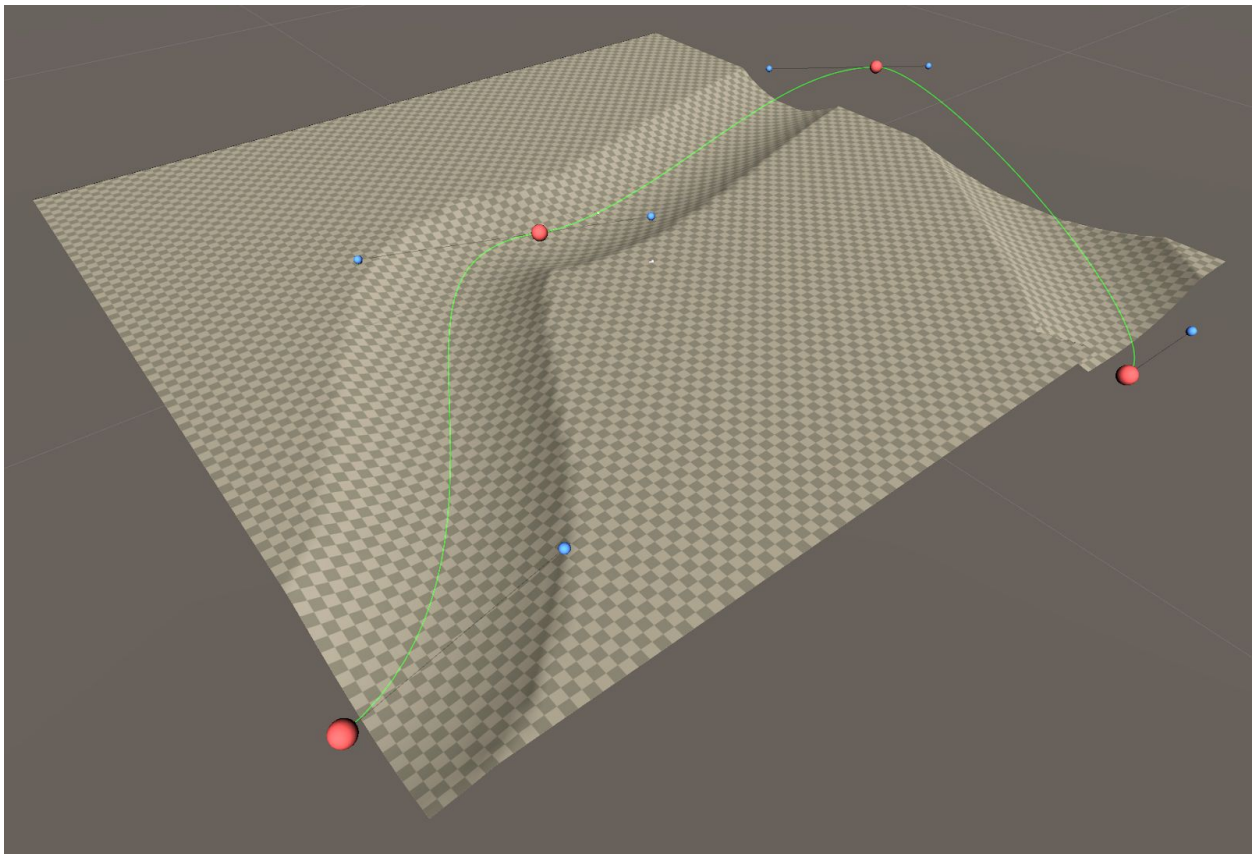
1. Raccolta valori dimensionali del componente Terrain.
   E' possibile gestire anche terreni di dimensioni diverse da 40m x 40m.
   Si consiglia di mantenere sempre un'area quadrata per facilitare la mappatura corretta dei punti del terreno all'interno delle texture di utilità.
2. Preparazione delle strutture dati che ospiteranno i campi vettoriali che si concretizzeranno nelle texture di utilità.
3. Si percorre la curva di Bézier dalla sua origine fino alla sua terminazione.
   Per ogni punto individuato si calcolano la direzione tangente e normale alla curva.
   Si percorre la direzione normale e anti-normale alla curva nel punto per una distanza pari a $RiverWidth / 2 + RiverBorderWidth$.
   Per ogni punto individuato sulla direzione normale e anti-normale si calcola la sua distanza dalla curva e la si mette a rapporto con la distanza massima percorribile in una delle due direzioni.
   Il valore ottenuto varia in un intervallo tra 0 e 1, ci permette quindi di campionare il parametro *Depth Curve* ed ottenere i valori da inserire nel rispettivo pixel di depth-map e height-map.
   Per popolare la flow-map inseriamo nei pixel associati nei punti individuati sulla direzione normale e anti-normale il colore corrispondente alla direzione tangente alla curva.

4. Vengono passati i valori della height-map al componente Terrain forzandolo ad aggiornare la sua morfologia nell'ambiente 3D.
   Il passaggio di questi dati avviene attraverso il metodo della classe *TerrainData SetHeights.*
5. Si serializzano le texture di utilità ai percorsi specificati come parametri di esecuzione dello script.

Una volta eseguito lo script il Terrain component ad esso associato risulterà modificato secondo i parametri da noi inseriti.
Di seguito sono mostrate le immagini catturate su una scena di test avente un terreno di dimensione 100mx100m sul quale viene eseguito lo script parametrizzato con una curva complessa. Sono riportate anche le texture di utilità esportate dallo script a fine esecuzione.



Scena prima dell'esecuzione dello script.
è ben visibile come il terreno sia totalmente pianeggiante.



Scena risultante dall'esecuzione dello script.
È evidente il letto del fiume scavato in corrispondenza della curva di Bézier.

Texture di utilità esportate dall'esecuzione dello script.
In ordine sono riportate la flow-map, la height-map e la depth-map.

Da un'analisi approfondita delle texture di utilità esportate emerge che nei punti in cui la curva varia rapidamente la sua direzione, nel caso di terreni molto estesi, si forma sulle texture un effetto *seghettato*. Questo artefatto è dovuto ad una frequenza di campionatura dei punti troppo bassa.

Per ridurre questi artefatti è possibile aumentare la frequenza di campionamento andando a modificare la variabile locale *deltaTime* all'interno del metodo *GenerateRiver()* dello script con una frequenza maggiore (di default ha una frequenza di una campionamento di 1 su 30m).
Aumentare la frequenza di campionamento richiede tempi di esecuzione dello script maggiori.
Un'altra soluzione possibile per mitigare la presenza di questi artefatti è l'utilizzo dell'effetto "Gaussian Blur" [6] tramite programmi esterni per la modifica delle immagini.

3.1.3 Vegetazione e dettagli

Generata la geometria, sono stati usati gli strumenti integrati nel componente Terrain nativo di Unity per aggiungere le texture e l'erba al terreno.
Per aggiungere le texture è stato usato lo strumento "Paint Texture" del componente Terrain e successivamente utilizzando lo strumento "Paint Details" è stata inserita l'erba.
Per aggiungere alla scena gli alberi e le rocce è stato deciso di non utilizzare gli strumenti offerti dal componente Terrain ma di inserirli nella scena direttamente come oggetti indipendenti. Questa scelta è stata effettuata per avere maggiore controllo sul singolo oggetto senza dover passare attraverso il componente Terrain.
All'interno della scena è presente anche il componente *WindZone*. Questo componente interagisce con la vegetazione e la anima facendola muovere come fosse sottoposta ad una lieve brezza.

3.1.4 Illuminazione

All'interno della scena è presente un'unica luce direzionale che emula l'illuminazione solare. La luce è stata implementata attraverso il componente nativo di Unity *Light* agganciato all'oggetto di scena di nome *Sun Directional Light* ed è stata impostata per far emettere le ombre agli oggetti di scena. Le ombre emesse sono calcolate a tempo di esecuzione, questa opzione rende più fedele la resa grafica a scapito di un maggiore costo computazionale.

3.1.5 Geometria del fiume

Il fiume è implementato attraverso la geometria di un semplice quadrato sopra il quale è applicato il materiale *RFSBWater*. L'oggetto di scena impiegato come geometria dell'acqua è l'oggetto *Water Mesh*. Questo oggetto non ha alcun vertice in movimento nella sua geometria, tutto l'effetto di moto è dato dall'apposito materiale ad esso applicato.

3.2 Shader dell’acqua

L’acqua è uno dei materiali più complessi da rappresentare accuratamente a causa delle sue particolari caratteristiche fisiche. L’acqua, infatti, varia molto il suo “comportamento visivo” a seconda della situazione in cui la osserviamo. L’aspetto di un piccolo fiume è molto diverso rispetto a quello di una grande superficie d’acqua come un oceano.
Nello sviluppo di shader generalmente non ci si preoccupa di creare un materiale valido per tutte le casistiche d’uso, piuttosto ci si concentra su ognuna con una implementazione specifica.
Nel caso di questo applicativo mi sono concentrato nel creare uno shader valido per un fiume di media dimensione.
Ho preso come riferimento per l’aspetto del corso d’acqua da sintetizzare il canale del Reno presso Bologna (di seguito) attraverso alcune fotografie scattate il 12 Novembre 2019 e alcuni video reali sui quali era stato utilizzato l’algoritmo di OTV.

Foto del canale del Reno, presso Bologna

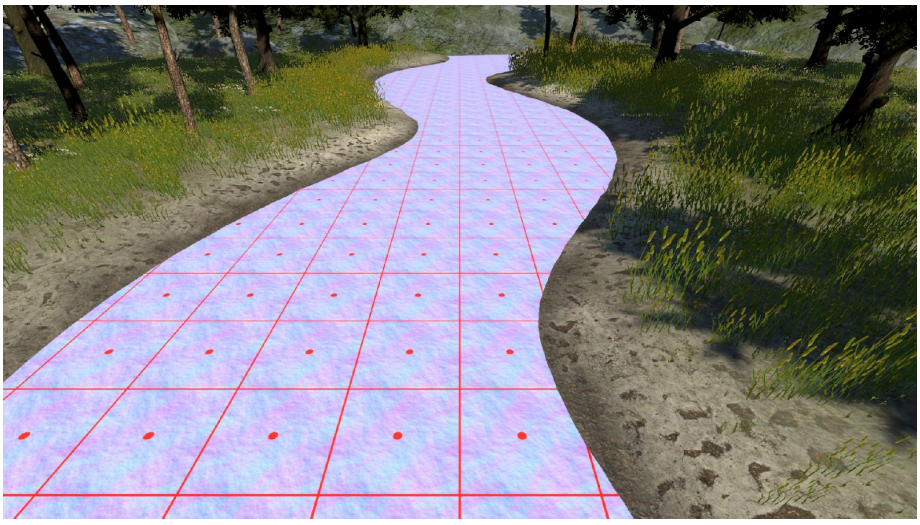
3.2.1 Generalità sull'implementazione

Lo shader dell'acqua è implementato attraverso l'uso della programmazione shader visuale "shader-graph" di Unity.
Lo shader è composto da un numerosi nodi, ognuno dei quali può essere visto come una funzione con input e output. I risultati di questi nodi andranno poi a confluire in un nodo di output che definirà l'aspetto finale delle superfici a cui è assegnato un materiale derivante dallo shader in questione.
Lo shader è stato implementato sul modello di "physically based rendering" (PBR) e come tale avrà come nodo di output tutta una serie di caratteristiche fisiche del materiale che serviranno al motore grafico per calcolare l'aspetto finale.

In questa tesi non sarà analizzata la totalità del comportamento dello shader, ci si concentrerà ad analizzare le sue due parti fondamentali, ovvero la programmazione del movimento delle increspature e quella della trasparenza nelle varie posizioni del fiume.
Il file dello shader è disponibile all'interno del progetto del framework al percorso:
Assets > Shaders > RFSBWater.shadergraph

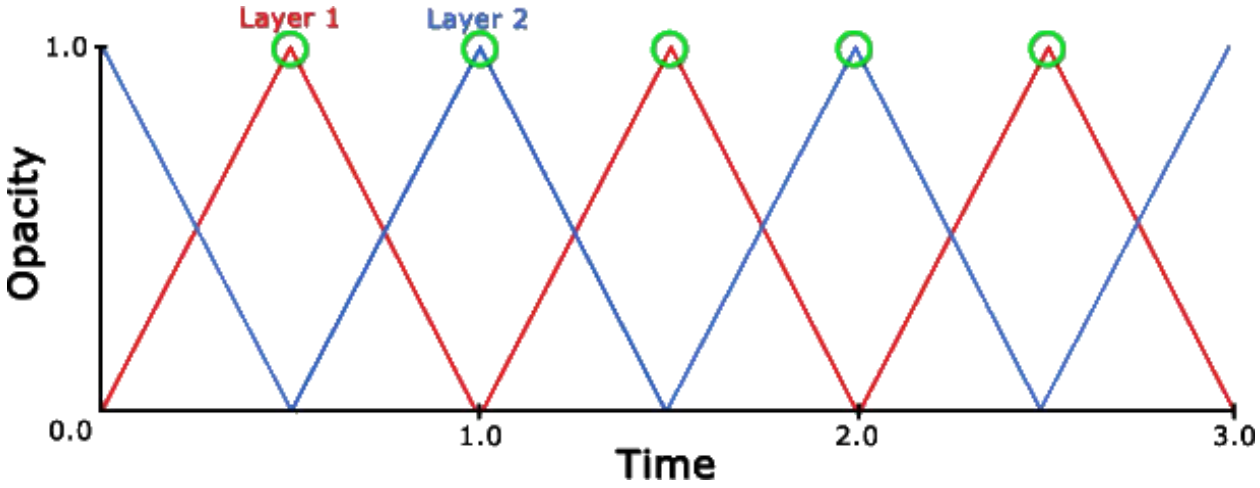
3.2.2 Movimento

Per sviluppare il movimento dell'acqua e delle sue increspature mi sono ispirato a una presentazione tenuta il 28 luglio 2010 al SIGGRAPH 2010 da Alex Vlachos [7], allora principal software architect presso Valve.
Le increspature sul flusso dell'acqua sono ottenute tramite l'utilizzo di normal-map [8] applicate in modo ripetuto e scalato ("tiling") sulla mesh del corso d'acqua.



Sopra è riportata un'immagine della scena in è stato reso visibile il "tiling" delle normal-map.

Per ottenere un effetto maggiormente realistico del movimento delle increspature e per evitare artefatti dovuti al tiling, generalmente si sommano tra loro vari livelli di normal-map aventi una diversa scala di “tiling” e un offset.
Nel caso di questa implementazione lo shader possiede 2 livelli di normal-map applicati sull’intera superficie del fiume.
L'opacità dei livelli di normal-map viene fatta variare nel tempo secondo una funzione a dente di sega in modo da alternare i vari livelli.

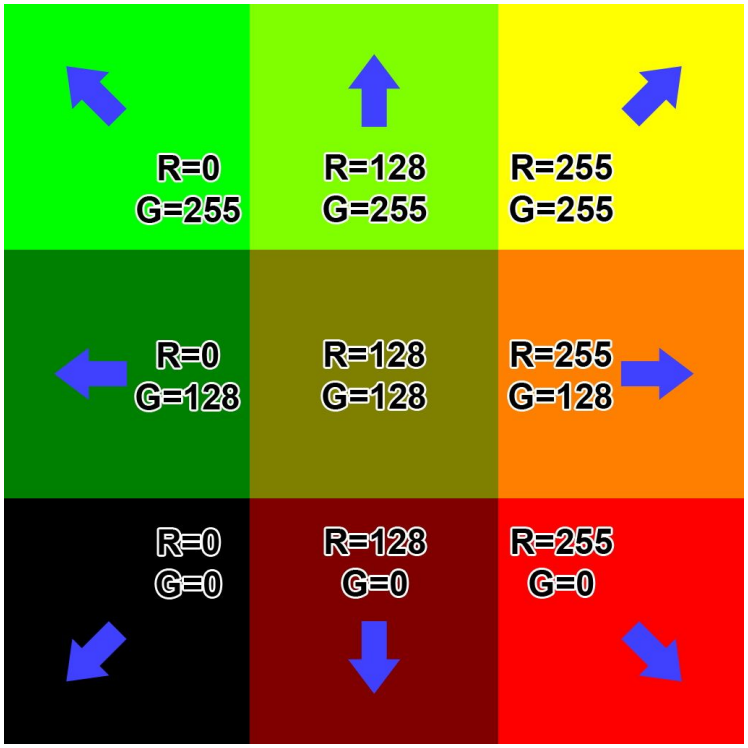


Funzione a dente di sega che mostra l’andamento nel tempo dell'opacità dei due livelli di normal-map (immagine presa dalla presentazione di Alex Vlachos sopracitata [7])

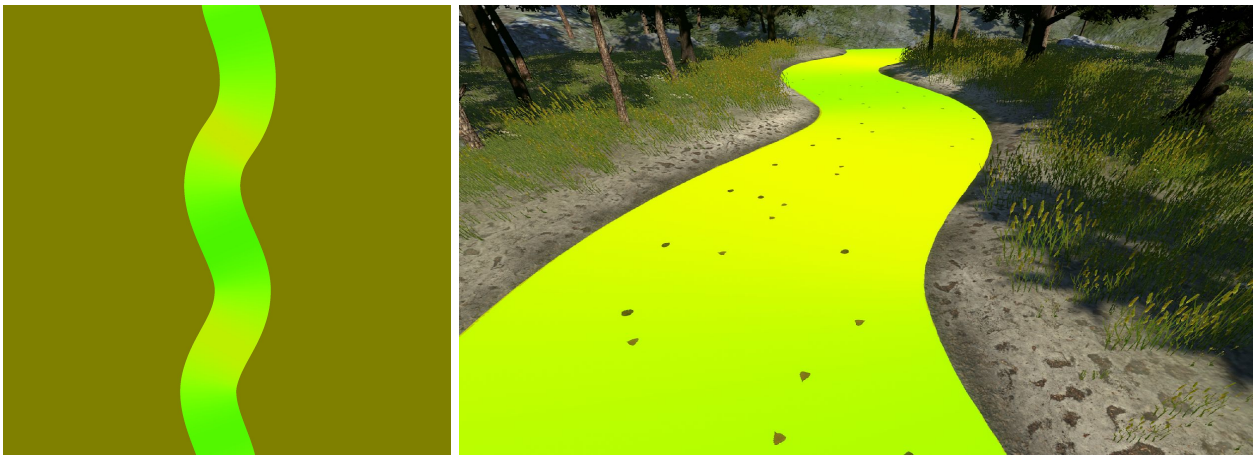
L’illusione del movimento dell’acqua è dato dalla distorsione nel tempo delle normal-map nella direzione del flusso.
La direzione dei flussi d’acqua viene mappata all’interno di una texture detta flow-map. Essa può essere considerata come la “fotografia” di un campo vettoriale in 2 dimensioni. Ogni pixel di una flow-map rappresenta un vettore, esso viene mappato all’interno delle componenti rosso (R) e verde (G) del colore del pixel (RGB) come segue:

$$v_x = (pixel.r - 0.5) * 2$$
$$v_y = (pixel.g - 0.5) * 2$$

La formula riportata considera le componenti di colore di un pixel come valori di tipo float variabili nell’intervallo 0-1.
Questa rappresentazione si distacca dalla più generale rappresentazione dei colori in formato RGB 24 bit, dove ad ogni canale di colore sono assegnati 8 bit, ovvero un intervallo da 0 a 255.



Questa legenda mette in evidenza come vengono codificate le informazioni vettoriali all'interno dei canali R e G di una flow-map avente formato di colore RGB 24 bit.



A sinistra la texture utilizzata come flow-map, a destra la stessa applicata al piano dell'acqua della scena.

All'interno del progetto Unity la gestione del movimento dell'acqua per lo shader dell'acqua è ritrovabile nello shader "subgraph" al percorso:
Assets > Shaders > Subgraphs > Flowmap.shadersubgraph

3.2.3 Trasparenza

La trasparenza dell'acqua viene ottenuta dallo shader attraverso l'utilizzo di una texture che ricopre il ruolo di mappa dell'opacità dell'acqua nei vari punti della superficie. La mappa è formata da gradazioni di grigio, ovvero sui canali di colore RGB dell'immagine abbiamo valori tra loro uguali che variano nell'intervallo [0,1]. Al valore bianco (R:1, G:1, B:1) faremo corrispondere un punto a trasparenza massima mentre al valore nero (R:0, G:0, B:0) corrisponderà un punto ad opacità massima.

A sinistra la texture utilizzata come maschera di trasparenza, a destra la medesima applicata al piano dell'acqua della scena.

Questa tecnica ci permette di emulare il comportamento visivo di un fiume reale avente una alta torbidità verso il centro del suo corso che va man mano a ridursi più ci si avvicina alle sponde.
I valori estratti da questa texture [0,1] sono poi rimappati su una scala di opacità dove lo 0 corrisponde all'opacità minima (trasparenza massima).
Una volta ottenuti i valori di opacità si possono già fornire in output allo shader nel valore di "alpha".

3.2.4 Riflessi

I riflessi sulla superficie dell'acqua sono ottenuti campionando una cubemap. Essa è uno speciale tipo di texture su cui viene mappato un paesaggio.
Un modo semplice e computazionalmente poco costoso per ottenere dei riflessi sulla superficie dell'acqua è quello di campionare la cubemap e applicare i colori campionati sulla superficie riflettente.
I riflessi ottenuti non sono reali, ma il risultato visivo finale e il basso costo computazionale rendono questa tecnica ideale per i requisiti grafici di questo applicativo.

3.2.5 Risultato finale

In questa immagine è riportata l'immagine della scena di benchmark in cui è visibile la resa finale del materiale dell'acqua applicato alla geometria del fiume.

3.2.6 Possibili miglioramenti

La resa grafica del materiale dell'acqua in fase di revisione è stata giudicata buona e ho potuto procedere allo sviluppo degli elementi rimanenti della scena.
Rimangono tuttavia alcune problematiche grafiche che ad un occhio esperto sono visibili e potrebbero essere risolte in una prossima iterazione di sviluppo.
La principale problematica visibile è quella del "tiling" delle normal-map. Malgrado l'utilizzo di un offset, di diverse scale di "tiling" e l'applicazione di "rumore", rimane comunque visibile la ripetizione delle increspature.
Questo problema è risolvibile utilizzando alcuni algoritmi che puntano a randomizzare il mapping delle texture sulle superfici delle geometrie. Per futuri sviluppi in questo senso, è allegata alla tesi il il riferimento [9], corrispondente alla pagina del blog personale dello sviluppatore grafico Inigo Quilez in cui vengono spiegati alcuni algoritmi utili a eliminare gli effetti di ripetizione delle texture dovute al "tiling".

3.3 Movimento particelle

Il requisito fondamentale dell'applicativo è quello di avere una scena fluviale con dei traccianti che scorrono a direzione e velocità conosciuta in modo da poter misurare la qualità degli algoritmi di OTV. Pertanto è cruciale avere un sistema flessibile che permetta di gestire la scena fluviale e il movimento delle particelle sul corso del fiume.

3.3.1 Generalità sull'implementazione

All'interno del progetto dell'applicativo, tre componenti svolgono un ruolo chiave nella gestione delle proprietà del fiume e nel controllo delle particelle che vi scorrono sopra. Questi componenti sono:

- *WaterParticle*, script che si occupa di gestire il movimento e il ciclo di vita delle particelle una volta generate.
- *WaterParticlesEmitter*, script che si occupa di creare le particelle in un determinato punto del fiume.
- *WaterSystemController*, script che si occupa di fare da controller del sistema che gestisce le proprietà dell'acqua e degli oggetti traccianti. Esso è implementato come singleton. Il riferimento alla sua istanza è quindi accessibile a tutti gli script che necessitano accedervi.

Andiamo ora ad analizzare in dettaglio ogni singolo componente.

3.3.2 Water Particle

Il componente *WaterParticle* va a definire il comportamento di ogni singolo oggetto tracciante. Esso è strettamente dipendente dall'istanza del singleton WaterSystemController attraverso il quale determina la velocità, la direzione di movimento e se distruggere l'istanza dell'oggetto tracciante a cui è assegnato.
Ogni componente *WaterParticle* utilizza una strategia di "polling" per accedere alle informazioni sull'istanza del *WaterSystemController* presente in scena.
Queste operazioni sono svolte all'interno del proprio ciclo di *Update()*, vengono quindi eseguite per ogni frame renderizzato.
Ogni istanza del componente *WaterParticle* è dotato di un ID univoco che gli viene assegnato dal *WaterSystemController* al momento della sua creazione. L'ID è necessario per differenziare all'interno del log esportato dal benchmark i dati relativi ad ogni oggetto tracciante.

3.3.3 Water Particles Emitter

Il componente *WaterParticlesEmitter* implementa il pattern *"factory"*. Esso si occupa di creare gli oggetti che rappresenteranno i traccianti.
Questo componente necessita di alcuni parametri per funzionare correttamente. Essi devono essere assegnati tramite l'editor di Unity e non possono essere modificati a tempo di esecuzione. I parametri richiesti sono i seguenti:

- *Path*, riferimento ad una istanza di un componente *PathCreator*, ovvero una istanza di una curva di Bézier.
- *Path Position*, valore numerico compreso tra 0 e 1, definisce il punto sulla curva di Bézier dove verrà calcolata la normale alla curva su cui giaceranno i punti di generazione dei traccianti.
- *Width*, valore numerico positivo, definisce la larghezza dell'intervallo di punti utili alla generazione dei traccianti.
- *Particles prefabs*, lista di *prefabs* [10] dei traccianti.
  Ogni singolo prefab può essere visto come un modello da cui vengono generate copie. In questo caso abbiamo la lista di modelli di oggetti traccianti da cui, ad ogni creazione, viene selezionato randomicamente un oggetto che funge da modello da cui generare una copia. Aggiungendo prefabs diversi a questa lista si aumenta la varietà degli oggetti traccianti creati.

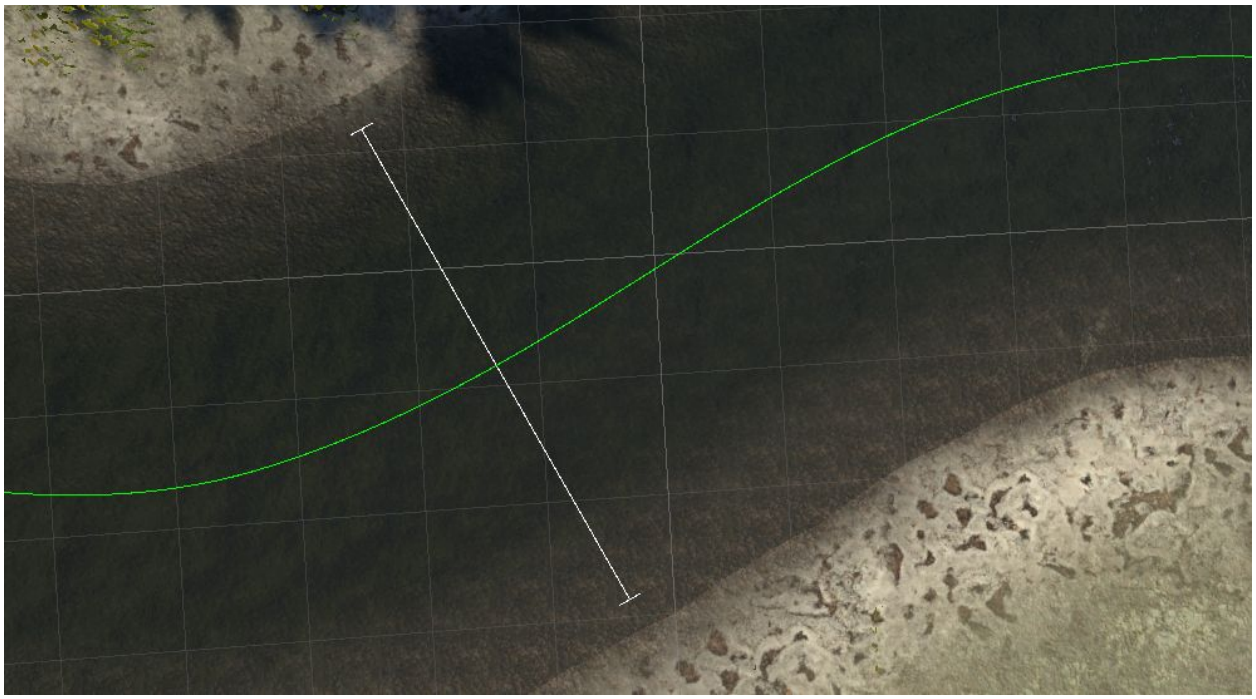


Immagine dell'editor di Unity in cui è mostrata, tramite linee di debug, la posizione dell'intervallo di generazione delle particelle rispetto alla curva di Bézier del corso del fiume.

Il componente *WaterParticlesEmitter* è privo di ciclo Update(), esso per operare richiede l'invocazione esterna del metodo *factory CreateParticle(int id)*. Questo metodo si occupa di creare un nuovo tracciante partendo dal modello di uno dei prefab della lista *Particles prefabs* estratto in modo causale. Successivamente gli applica una scala e una rotazione attorno all'asse *Y* randomica.
Una volta creato l'oggetto del tracciante gli aggancia il componente *WaterParticle* e gli assegna l'id passato come parametro. Il riferimento all'istanza del componente *WaterParticle* viene poi passato come valore di ritorno del metodo *factory*. Nella scena questo componente è posseduto dall'oggetto *RFSB Partic Emitter*.

3.3.4 Water System Controller

Il *WaterSystemController* è il cuore della scena di benchmark. E' implementato secondo il pattern singleton per facilitare la gestione dei riferimenti ai componenti che devono interagirvi.
Il componente *WaterSystemController* si occupa delle seguenti funzioni:

- Gestione dei parametri principali del corso d'acqua.
  Questo componente si occupa di mantenere in memoria i parametri della velocità del corso d'acqua, la sua flow-map del corso d'acqua e il parametro che definisce ogni quanti secondi creare un nuovo oggetto tracciante sul fiume.
- Settaggio parametri del materiale dell'acqua.
  Lo script *WaterSystemController* si occupa di mantenere un riferimento all'oggetto che possiede la geometria (mesh) del corso d'acqua in modo da poter aggiornare i parametri del suo materiale in caso di modifiche.
- Gestione della creazione degli oggetti traccianti.
  Questa funzione viene svolta all'interno del loop *Update()* del componente. Ad ogni frame lo script controlla se è passato sufficiente tempo dalla creazione della particella precedente. In caso la condizione sia soddisfatta si usa il riferimento al *WaterParticlesEmitter* per creare un nuovo oggetto tracciante. Appena creato l'oggetto tracciante viene poi registrato all'interno di una mappa che serve a tenere traccia degli oggetti traccianti attivi presenti in scena.
- Gestione del ciclo di vita degli oggetti traccianti (*WaterParticle*).
  Gli oggetti traccianti accedono ad ogni frame all'istanza singleton *WaterSystemController* attraverso il loro componente *WaterParticle* per determinare la direzione, il verso e la velocità del quale devono muoversi. Questa operazione viene svolta dal *WaterSystemController* tramite l'uso di una flow-map che permette di risalire dalla posizione delle singole particelle nella scena ad un vettore come spiegato al paragrafo 3.2.3. Oltre a queste funzioni, ogni componente *WaterParticle chiede all'interno del ciclo di Update()* all'istanza

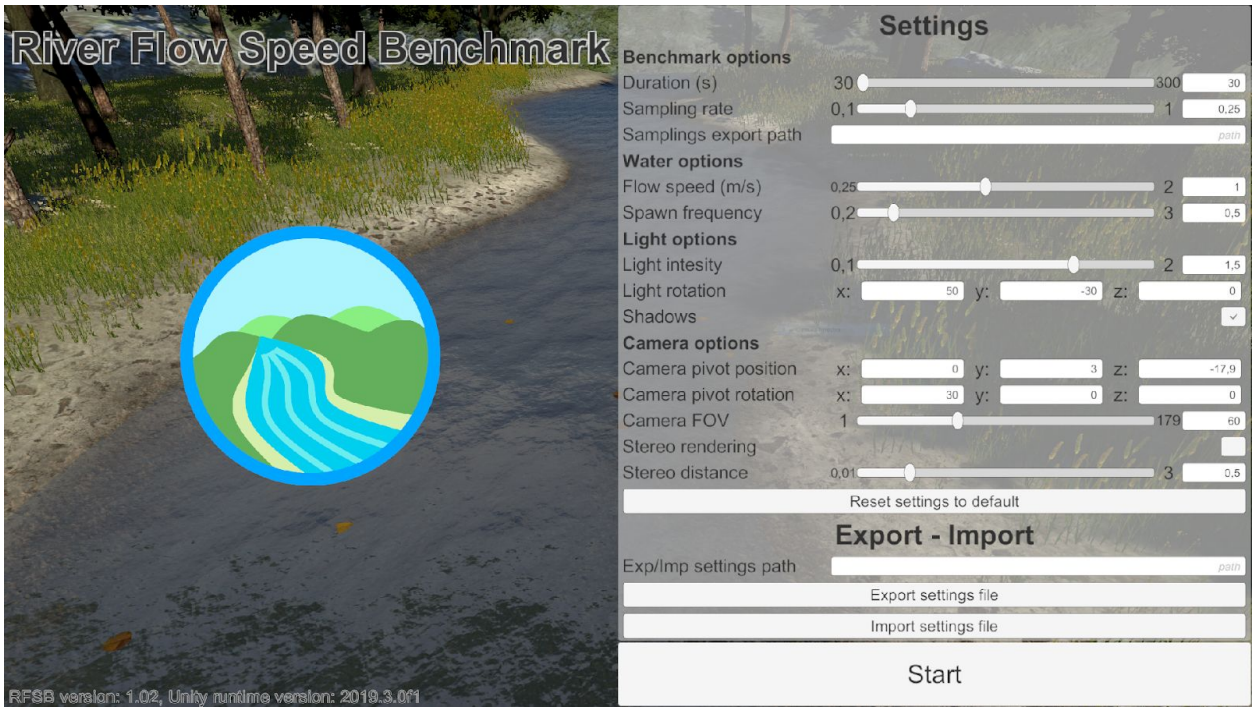
del *WaterSystemController* di determinare se la loro posizione è al di fuori della scena fluviale. In caso affermativo viene chiamata la rimozione dalla mappa dei traccianti attivi in scena e viene distrutto l'oggetto tracciante.

- Mappatura degli oggetti traccianti presenti in scena.
  Questa funzione è particolarmente utile per lo scopo finale del benchmark; permette infatti di avere in ogni istante un riferimento agli oggetti traccianti presenti in scena. Questo ci dà la possibilità di risalire alle loro informazioni come identificativo, posizione nella scena, direzione e velocità di movimento.
  La raccolta di questi dati a periodi costanti di tempo è uno strumento che può aiutare gli utilizzatori del benchmark a valutare meglio la qualità degli algoritmi di OTV testati. Queste informazioni possono essere ottenute dall'istanza *WaterSystemController* attraverso la chiamata del metodo pubblico *ExportParticlesData()*. Esso restituisce un array di oggetti *WaterParticleData*, uno per ogni tracciante in scena.

Nella scena di benchmark l'oggetto che possiede l'istanza singleton di questo componente è quello denominato *RFSB Water System*.

3.4 Interfaccia utente

L'interfaccia utente (UI) è stato uno dei punti fondamentali del lavoro sull'applicativo di benchmark. Questa attenzione è derivata dal voler progettare un ambiente quanto più facile da usare e con una modalità comoda e veloce per modificare tutte le opzioni disponibili all'interno dell'applicativo.
La UI era il punto debole della prima versione dell'applicativo sviluppato durante il tirocinio universitario [2] in quanto non permetteva di accedere in modo diretto a tutte le opzioni del software. Ad esempio, per modificare la posizione o la rotazione delle camere che si occupavano del rendering della scena era necessario accedere al file delle impostazioni che veniva generato all'interno della cartella in cui veniva installato l'applicativo.
La nuova versione dell'applicativo ha un'interfaccia migliorata che dà la possibilità di esportare ed importare i settaggi sotto forma di file Json. Questo permette di generare e caricare profili diversi in fase di utilizzo dell'applicativo.



Schermata principale della versione 1.02 dell'applicativo RFSB nella quale è visibile il pannello laterale con le opzioni del benchmark. Ogni opzione una volta modificata mostra il suo risultato a tempo di esecuzione; questo facilita l'utente finale nella creazione di scene utili per mettere alla prova gli algoritmi di OTV.

3.4.1 Generalità sull'implementazione

L'interfaccia grafica dell'applicativo è stata sviluppata completamente attraverso l'utilizzo dei componenti nativi di Unity [11]. Essi sono raccolti all'interno del namespace *UnityEngine.UI*. All'interno dell'applicativo tutte le interfacce grafiche hanno come unico ricevitore degli eventi lanciati l'istanza del componente singleton *ApplicationController*.
Questo componente è di fatto il controller dell'applicazione e si occupa delle seguenti funzioni:

- Gestione dei settaggi dell'applicativo.
  Riceve un evento quando un'opzione viene aggiornata tramite l'interfaccia grafica e si occupa di andare a salvarne il valore all'interno del campo specifico dell'oggetto *settings* di classe *BenchmarkSettingsData*.
  Il componente *ApplicationController* non riceve questi eventi direttamente dalla UI ma ogni volta che il valore di una opzione viene modificato. A lanciare questi eventi sono le istanze delle singole opzioni sotto forma di speciali *ScriptableObjects* [12].
  Questo sistema è abbastanza complesso e sarà approfondito meglio nel prossimo paragrafo con un esempio pratico di implementazione.
  Una volta che i settaggi sono correttamente salvati vengono passati alle istanze delle classi che dovranno utilizzare tali opzioni. Ad esempio, il valore aggiornato della velocità dell'acqua verrà passato all'istanza di scena del *WaterSystemController*.
- Gestione del ciclo di vita del benchmark.
  Il componente *ApplicationController* è il controllore del benchmark. Esso riceve dall'interfaccia grafica gli eventi di avvio e terminazione forzata del benchmark. In caso di evento di avvio del benchmark, il componente si occupa di inizializzare i settaggi ai componenti coinvolti e avvia la coroutine [12] definita dal metodo *BenchmarkLoop()*. Questa logica di inizializzazione si trova all'interno del metodo *StartBenchmark()*. All'interno di questo metodo viene svolta la logica utile alla generazione del file di log. Al termine della durata del benchmark viene invocato il metodo *EndBenchmark()*. Questo metodo si occupa principalmente di terminare la coroutine *BenchmarkLoop()* e di esportare il file di log. Il metodo *EndBenchmark()* è lo stesso che viene invocato dall'evento dell'interfaccia grafica che richiede l'interruzione del benchmark prima del suo completamento.
- Serializzazione e deserializzazione delle impostazioni.
  L'interfaccia grafica dell'applicativo mette a disposizione la possibilità di esportare e importare sotto forma di file i settaggi dell'applicativo. La logica di queste operazioni è implementata all'interno dell'applicativo tramite l'uso di

oggetti che implementano le interfacce *IBenchmarkSettingsDataReader* e *IBenchmarkSettingsDataWriter*. Queste ultime definiscono i metodi per andare a serializzare/deserializzare oggetti della classe *BenchmarkSettingsData*.

- Attivazione e disattivazioni pannelli dell'interfaccia grafica.
  Nell'applicativo sono presenti 2 pannelli principali che vengono alternati a seconda che l'applicativo abbia o meno una sessione di benchmark attiva. I due pannelli sono ritrovabili rispettivamente negli oggetti *Settings Interface* e *Benchmark Interface*. Questi ultimi non sono mai attivi contemporaneamente in scena e sono gli oggetti radice di tutti gli altri elementi di interfaccia delle rispettive situazioni. All'interno di *Settings Interface* troveremo tutti gli oggetti che andranno a formare la schermata del titolo e dei settaggi (quella che incontriamo al primo avvio del programma).
  L'oggetto *Benchmark Interface* contiene invece l'interfaccia visibile una volta in corso una sessione di benchmark.

3.4.2 MyType

Nel paragrafo 3.4.1, abbiamo introdotto come ogni opzione presente nell'interfaccia sia implementata attraverso una specifica istanza di uno speciale *ScriptableObject* [12].
Non andando ad approfondire il funzionamento degli *ScriptableObject*, mi limito a descriverli come contenitori di dati la cui istanza può essere salvata sotto forma di asset tramite l'editor di Unity. Il riferimento all'istanza può essere mantenuta tra più componenti.
Gli *ScriptableObject* possono contenere anche logica applicativa.

La classe *MyType* (namespace *RFSB.UI.Data*) estende la classe *ScriptableObject* di Unity e incapsula un tipo generico con i suoi getters e setters ed espone l'evento *OnValueChanged*.
I componenti esterni che possiedono un riferimento ad una istanza derivante da *MyType* possono quindi iscriversi all'evento *OnValueChanged* e aggiornarsi quando il valore contenuto all'interno dell'istanza viene settato tramite il metodo *Set*.
Quando il valore viene cambiato attraverso il metodo *SetSilent* non è invocato l'evento. Questa logica "silente" è pensata per gestire casi di immissioni non valide in cui è necessario resettare queste variabili senza notificare i componenti iscritti all'evento.

All'interno dell'applicativo sono stati implementati alcuni tipi fondamentali come estensioni della classe *MyType* tra cui: *MyInt, MyFloat*, *MyBool, MyString* e *MyVector3*.

Per ognuno di queste classi derivate da *MyType*, aggiungendo l'attributo *CreateAssetMenu* [13] è stata resa possibile la creazione attraverso l'editor di Unity delle istanze sotto forma di asset di questi speciali *ScriptableObject*.

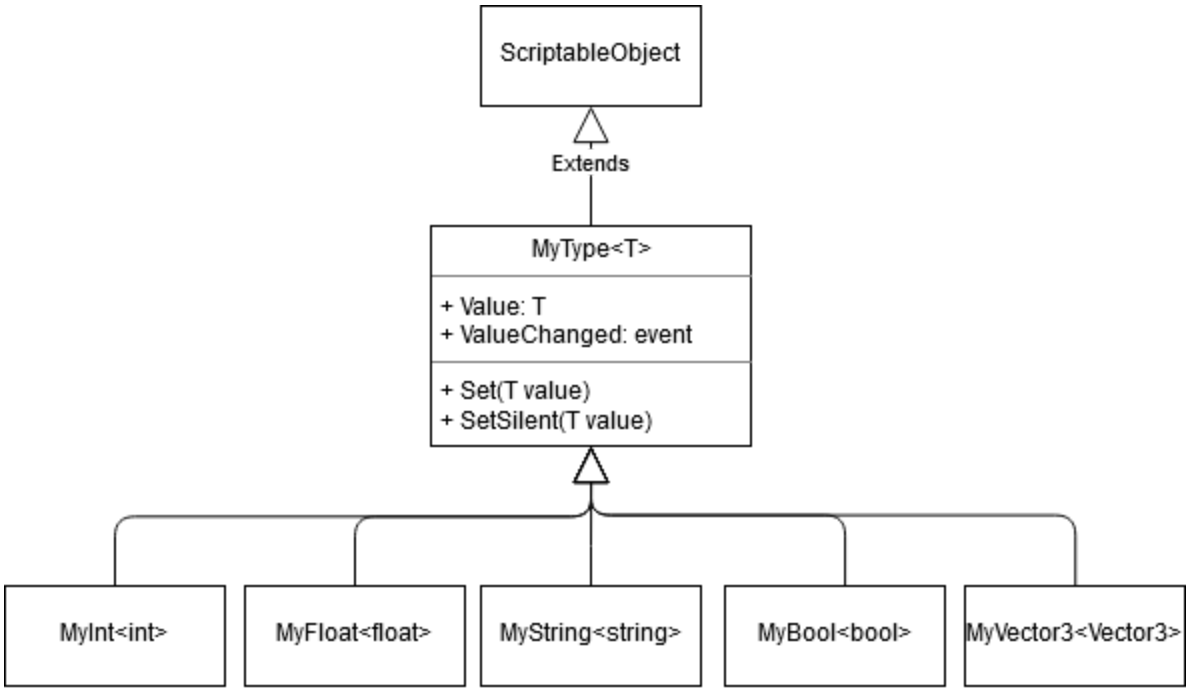


Diagramma UML in cui è visibile graficamente la relazione tra le classi descritte.

Per inserire nuove classi derivate da *MyType* è necessario creare un nuovo file *C#* all'interno del progetto e definire la nuova classe con il relativo tipo di dato che va ad incapsulare.

3.4.3 Inserimento di un nuovo settaggio

Andiamo ora ad approfondire questi meccanismi attraverso un esempio pratico. Ripercorriamo assieme le operazioni necessarie ad aggiungere all'applicativo l'opzione che specifica la rotazione dell'oggetto *Sun Directional Light*, ovvero la luce direzionale che rappresenta il sole in scena.

Il primo passo per aggiungere una opzione alla scena è quella di creare all'interno del nostro progetto un asset che rappresenti l'istanza della nostra opzione.
All'interno del progetto tutte le istanze degli *ScriptableObject* sono contenute nel percorso *Assets* > *UI Data*.
Per rappresentare una rotazione useremo una istanza dello *ScriptableObject MyVector3*, procediamo quindi ad aggiungere questa opzione tramite la top-bar di Unity selezionando *Assets* > *Create* > *MyVector3*.

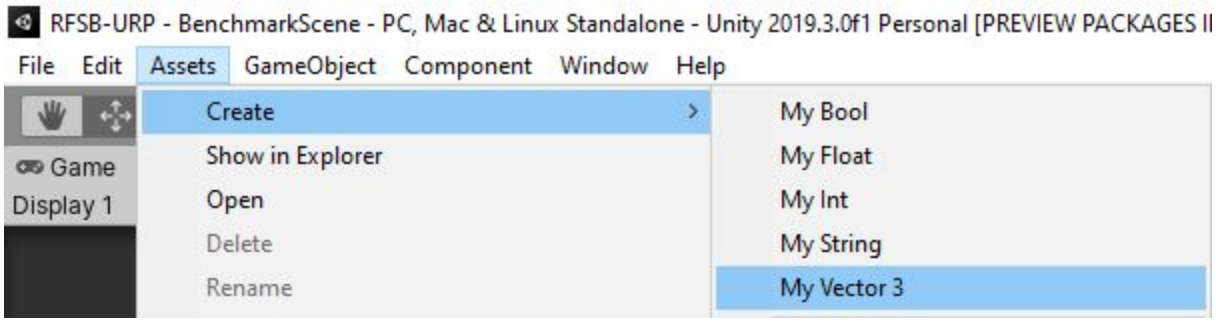


Immagine in cui è visibile il sotto-menu dell'editor di Unity da cui creare gli asset degli ScriptableObjects di tipo MyType all'interno del progetto.

Completata questa operazione, comparirà nella cartella selezionata un nuovo asset che rappresenta l'istanza condivisa della nostra opzione. E' consigliabile rinominare l'asset in modo che sia chiaro il suo scopo e il tipo di dato in esso contenuto.

Creato l'asset che contiene l'istanza della nostra opzione, è possibile passare all'implementazione all'interno dell'*ApplicationController* della logica che dovrà gestire la ricezione di eventi e la serializzazione/deserializzazione di questo settaggio all'interno dei file contenenti le impostazioni dell'applicativo.
In primis dobbiamo rendere possibile al componente *ApplicationController* il possesso di un riferimento all'asset della nostra opzione.
Aggiungiamo quindi alla lista di "asset-opzione" presente nello script un nuovo riferimento di tipo MyVector3 denominato *LightRotationSetting*.

Se vogliamo che la nuova opzione possa essere serializzata/deserializzata all'interno del file di settaggi dell'applicativo dobbiamo aggiungere un campo specifico per la nuova opzione all'interno della classe *BenchmarkSettingsData*.
Il componente *ApplicationController*, come accennato in precedenza, possiede una istanza di questa classe che utilizza come container unico dei settaggi dell'applicativo. Una volta che viene richiesto da parte dell'utente l'esportazione/importazione dei settaggi bisogna serializzare/deserializzare un oggetto della classe *BenchmarkSettingsData*.

Per migliorare la leggibilità del codice si consiglia di organizzare i settaggi in aree logiche. In questo caso è stato riportato il settaggio della rotazione della luce tra quelli che coinvolgono le altre proprietà della luce di scena.

Una volta aggiunto il campo per il riferimento all' "asset-opzione" è possibile passare alla registrazione e de-registrazione del componente *ApplicationController* all'evento *OnValueChanged* della nostra opzione sottoclasse di *MyType*.
L'operazione di registrazione è consigliabile effettuarla nel metodo *OnEnable()* dello script. Questo metodo viene ogni volta che l'oggetto in scena che possiede il componente, in questo caso l'*ApplicationController*, viene attivato. In caso di disattivazione viene eseguito il metodo *OnDisable()*, in esso andremo ad aggiungere la de-registrazione del componente all'evento.

All'interno del progetto è stato registrato all'evento *OnValueChanged* dell'opzione di rotazione della luce il metodo *LightRotationSetting_OnValueChanged()*.
All'interno di questo metodo è stata scritta la logica applicativa impiegata per aggiornare il valore dell'opzione all'interno dell'oggetto serializzabile contenente le opzioni dell'applicativo e ad aggiornare la rotazione della luce all'interno della scena.

Come ultima operazione all'interno dello script *ApplicationController* dobbiamo rendere possibile l'aggiornamento della visualizzazione di tutte le opzioni che sono de-serializzabili. Questa operazione viene svolta all'interno del metodo *UpdateSettingsUIValues()* e per renderlo compatibile con la nuova opzione della rotazione della luce basterà chiamare il metodo set sul riferimento dell' "asset-opzione" in questione passando il valore attualmente salvato nell'oggetto *settings* di classe *BenchmarkSettingsData*.
Con quest'ultima operazione lo script *ApplicationController* è completamente compatibile ad interagire con la nuova opzione inserita nell'applicativo.

Il prossimo passo è quello di creare l'interfaccia grafica dove l'utente potrà inserire i valori della nostra opzione.
In questa sezione, non mi soffermerò sulla creazione della UI attraverso degli strumenti nativi di Unity. Copierò l'oggetto *CamaraPivotRotationOption_Panel* rinominandolo in *LightRotationOption_Panel*.
All'interno della gerarchia di questo oggetto abbiamo un oggetto figlio denominato *Vector3InputField*, esso contiene il componente *Vector3InputField*. Quest'ultimo ha l'obiettivo di ricevere gli eventi generati dalla UI, di effettuare le operazioni di parsing dei valori immessi nei campi di testo e nel caso siano validi di settare il valore immesso sull'istanza "asset-opzione" di cui possiede un riferimento.
Vector3InputField è uno dei componenti di supporto all'interazione con la UI.
Durante lo sviluppo dell'applicativo ne sono stati sviluppati altri tra cui *SliderWithInputText*, *StringInputField* e *Toggle*.

L’ultima operazione che rimane per completare l’inserimento di questa opzione all’interno del programma è quella di passare ai componenti in scena coinvolti, in questo caso l’*ApplicationController* e il *Vector3InputField,* il riferimento all’ “asset-opzione” che controlla la rotazione della luce. Questo operazione è eseguibile con un semplice trascinamento dell’asset all’interno dello specifico campo dei componenti.

Con questa ultima operazione risulta completata l’aggiunta all’applicativo della nuova opzione per il controllo della direzione della luce della scena.

Sopra è riportato in ritaglio della schermata principale dell’applicativo in cui è visibile l’interfaccia della nuova opzione inserita per modificare la rotazione della luce di scena.

Come esposto, l’operazione di aggiunta di una opzione all’applicativo è molto veloce e richiede una minima modifica del codice nel caso l’opzione possa ri-utilizzare modelli di UI pre-esistenti in scena. Nel caso siano richieste implementazioni più complesse consiglio di approfondire il funzionamento del sistema di UI [11] e degli *ScriptableObjects* [12] di Unity e del componente *ApplicationController* dell’applicativo.

3.4.4 Opzioni disponibili

In questa ultima sezione dedicato all’interfaccia grafica dell’applicativo sarà descritta ogni opzione disponibile all’interno dell’interfaccia grafica e si andrà creare un piccolo manuale utile agli utilizzatori del benchmark.
Per ogni settaggio disponibile sarà indicato il nome con cui è salvata nel progetto lo specifico “asset-opzione”. Questa guida è aggiornata alla versione 1.02 dell’applicativo RFSB.

Le opzioni all’interno dell’applicativo sono organizzate in gruppi logici.
Di seguito riporto le opzioni legate al controllo della generazione dei video di benchmark:

- *Duration*, esprime in secondi la durata del benchmark.
  Questa opzione è incapsulata nell’“asset-opzione” *DurationUIField_Float*.
- *Sampling Rate*, esprime il sampling rate dei dati esportati nel log del benchmark.
  Questo valore descrive ogni quanti secondi il benchmark deve salvare i dati di

posizione, direzione, velocità ed identificativo degli oggetti traccianti del benchmark.
Questa opzione è incapsulata nell'"asset-opzione" *SamplingRateUIField_Float*.

- *Samplings export path*, questa opzione specifica il percorso della directory di sistema in cui verrà esportato il file Json di log del benchmark. Questa opzione è incapsulata nell'"asset-opzione" *SamplingsExportPathUIField_String*.

Di seguito sono riportate le opzioni legate al controllo dell'acqua e degli oggetti traccianti:

- *Flow speed*, esprime in metri al secondo la velocità media del corso d'acqua e degli oggetti traccianti. Questa opzione è incapsulata nell'"asset-opzione" *FlowSpeedUIField_Float*.
- *Spawn frequency*, esprime ogni quanti secondi viene generato un nuovo oggetto tracciante in scena. Questa opzione è incapsulata nell'"asset-opzione" *ParticlesSpawnFrequencyUIField_Float*.

Di seguito sono riportate le opzioni legate al controllo della luce di scena:

- *Light intensity*, esprime l'intensità della luce della scena. Questa opzione è incapsulata nell'"asset-opzione" *LightIntensityUIField_Float*.
- *Light rotation*, permette di controllare la rotazione della luce di scena. La rotazione è espressa sotto forma di angoli di Eulero. Questa opzione è incapsulata nell'"asset-opzione" *LightRotationUIField_Float*.
- *Shadows*, questa opzione determina se nella scena è attivo o meno il rendering delle ombre. Questa opzione permette sui computer meno potenti di risparmiare risorse in fase di rendering al costo di una resa grafica peggiore. Questa opzione è incapsulata nell'"asset-opzione" *ShadowsActiveUIField_Bool*.

Di seguito sono riportate le opzioni legate al controllo delle camere impiegate nel rendering della scena:

- *Camera pivot position*, definisce la posizione nella scena del pivot delle telecamere. Nel caso non sia attiva l'opzione del rendering stereo, la posizione specificata satà la posizione della camera, in caso contrario la posizione specificata sarà il centro tra le due camere stereo. Questa opzione è incapsulata nell'"asset-opzione" *CameraPivotPositionUIField_Vector3*.
- *Camera pivot rotation*, definisce la rotazione nella scena del pivot delle telecamere. Nel caso non sia attiva l'opzione del rendering stereo, la posizione specificata sarà la rotazione della camera, in caso contrario la rotazione specificata sarà quella del pivot delle due camere stereo. Questa opzione è incapsulata nell'"asset-opzione" *CameraPivotRotationUIField_Vector3*.

- *Camera FOV*, questa opzione definisce il Field Of View (campo visivo) di tutte le camere della scena. Questa opzione è incapsulata nell'"asset-opzione" *CameraFOVUIField_Float*.
- *Stereo rendering*, permette l'attivazione del rendering stereo. Questa opzione è incapsulata nell'"asset-opzione" *StereoRenderingActiveUIField_bool*.
- *Stereo distance*, definisce la distanza tra le due camere stereo. Al centro delle due camere si trova il pivot che controlla la loro posizione e rotazione. L'opzione è incapsulata nell'"asset-opzione" *StereoCamerasDistanceUIField_Float*.

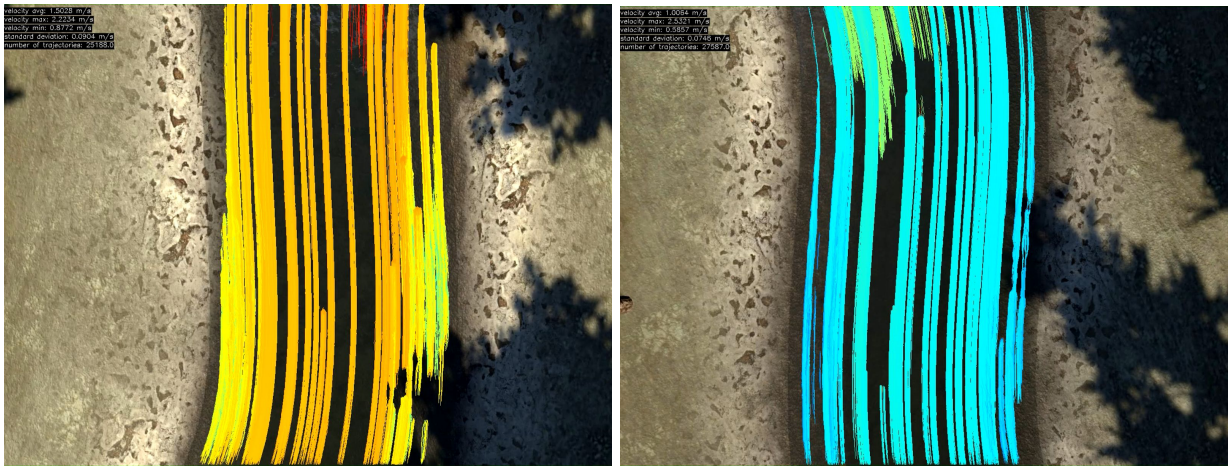
L'ultima sezione delle opzioni dell'applicativo include le funzioni di esportazione e importazione dei file Json contenente i settaggi del programma.
In questa sezione troviamo un unico campo denominato *Exp/Imp settings path*. Esso ha una funzione bivalente a seconda che venga invocata la funzione di esportazione o di importazione dei settaggi.
In caso di esportazione, l'applicativo si aspetta di trovare all'interno del campo *Exp/Imp settings path* un percorso ad una directory del sistema in cui andare a salvare il file Json contenente i settaggi dell'applicativo.
In caso di importazione, l'applicativo si aspetta di leggere all'interno del campo *Exp/Imp settings path* il percorso completo del file Json contenente i settaggi che si vogliono importare nell'applicativo.
L'opzione del campo *Exp/Imp settings path* è incapsulata all'interno dell'"asset-opzione" *ExportImportSettingsFilePathUIField_String*.

Risultati sperimentali

4.1 Test effettuati

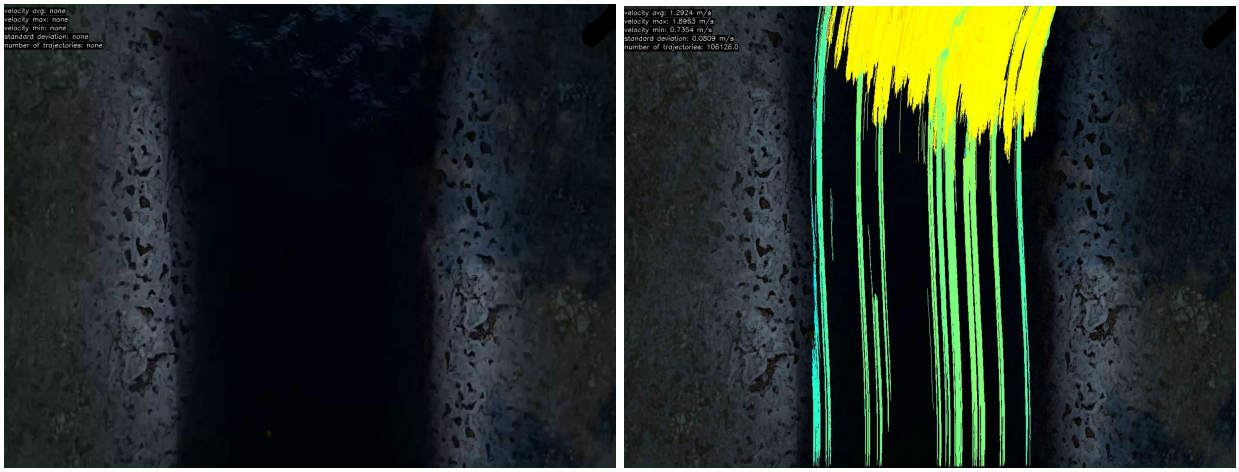
Il nuovo applicativo di benchmark è stato subito utilizzato per generare alcuni video sintetici da analizzare attraverso l'algoritmo di OTV proposto in [1]. Il test sui video generati è stato effettuato dal laureando Riccardo Turra come continuazione del lavoro svolto durante la scrittura della sua tesi di laurea [3].
Tale valutazione è consistita nella generazione di 12 video diversi attraverso l'utilizzo della nuova versione dell'applicativo RFSB (versione 1.01) e del programma OBS Studio per la registrazione dello schermo.
I video generati hanno tutti la direzione della camera ortogonale al corso d'acqua, ma variano nel posizionamento della camera nella scena, nell'altezza della camera rispetto al corso d'acqua, nell'intensità della luce "solare" e nella velocità dei traccianti sul corso dell'acqua.

Dal test effettuato sui video emerge che i risultati restituiti dagli algoritmi sono molto vicini ai valori di "ground-truth". Questo risultato é importante perché avvalora la qualità dell'algoritmo testato e mostra come i parametri della scena di benchmark possano considerarsi affidabili (a differenza della prima versione dell'applicativo in cui è stata richiesta una operazione di ingegneria inversa per risalire ai valori di "ground-truth").
Inoltre, dal test emerge che le scene generate tramite la nuova versione dell'applicativo hanno risultati meno suscettibili alla variazione dei parametri dell'algoritmo di OTV. In particolare, la variazione del parametro *final_min_distance* sembra influire in modo molto minore sui risultati finali rispetto alla prima versione della scena.



Frame tratti dai video 1 e 9 dove sono visibili le traiettorie individuate dall'algoritmo di OTV.

Un altro aspetto in cui la scena è migliorata e che trova conferma nelle misurazioni sperimentali è il tracciamento in condizioni di scarsa illuminazione.
Come riportato nella tesi di Riccardo Turra, nella versione precedente del benchmark gli oggetti traccianti sul corso del fiume risultavano avere un contrasto molto elevato rispetto alla scena, soprattutto in condizioni di scarsa illuminazione.
Questo li rendeva troppo facili da tracciare da parte dell'algoritmo di OTV, andando di fatto a rendere pressoché inutile la generazione di scene "notturne".
Nella nuova versione dell'applicativo, in condizioni di scarsa illuminazione l'algoritmo si trova in difficoltà e il numero di traccianti individuati diminuisce. Inoltre, in presenza di zone d'acqua con dei riflessi l'algoritmo individua il movimento delle increspature dell'acqua come traccianti.
Le misurazioni effettuate in queste condizioni risultano quindi distaccarsi molto dai valori di "ground-truth". Questi risultati sono però perfettamente coerenti con misurazioni di scene reali in presenza di scarsa illuminazione.



A sinistra il frame iniziale del video 5 dove è visibile il riflesso delle increspature dell'acqua.
A destra un frame tratto da un momento centrale del video 5 in cui è visibile la difficoltà che ha l'algoritmo di OTV testato ad identificare gli oggetti traccianti e come le increspature illuminate in movimento vengano conteggiate come traccianti.

Il test ha mostrato come in tutte le condizioni di luce venga considerato come tracciante il movimento delle increspature dell'acqua. Questo fenomeno è più accentuato in presenza di riflessi di luce che contrastano il colore delle increspature da quello della scena circostante (vedasi il caso del video 5 riportato in precedenza).
Quanto appena evidenziato non è da considerarsi totalmente un malus in quanto genera una difficoltà per gli algoritmi di OTV che è possibile si presenti in natura.
Nel caso di condizioni di luce diurna (come i video 1 e 9) l'algoritmo di OTV testato traccia solo inizialmente le traiettorie dovute alle increspature dell'acqua. Dopo la comparsa nel video di alcuni traccianti le increspature sembrano essere scartate dalla

misurazione. Questo fenomeno, in casi di illuminazione non critica, porta ad una sovrastima della velocità di circa il 10% durante i primi secondi della misurazione. Tuttavia, appena si aggiungono altre traiettorie in scena, la media della velocità si riallinea velocemente verso il valore di “ground-truth”.

Frame iniziale del video 1 in cui sono ben visibili le traiettorie calcolate a partire dal movimento delle increspature dell’acqua. è visibile come in queste prime fasi la misurazione si discosta di 0.14 m/s circa dal valore di “ground-truth” della velocità (1 m/s).

In conclusione, la nuova scena sembra confermare il corretto tracciamento da parte dell’algoritmo di OTV anche di traiettorie non rette.
Il fatto che il corso del fiume segua una forma sinusoidale non sembra mettere in difficoltà l’algoritmo di OTV, tuttavia bisognerebbe verificare questa affermazione tramite test più approfonditi e mirati a verificarne il corretto funzionamento anche con traiettorie più complesse sebbene esse non rappresentino un caso applicativo di particolare interesse nelle applicazioni per le quale l’algoritmo è stato proposto.

Conclusioni

5.1 Obiettivi raggiunti

La nuova versione dell'applicativo può essere considerata un miglioramento sostanziale rispetto alla precedente.

Il progetto è basato sull'ultima versione disponibile del motore grafico e sulla nuova Universal Render Pipeline. Quest'ultima, rende di fatto il progetto proiettato verso il futuro in quanto Unity ha in programma di sostituire la vecchia pipeline di rendering con la nuova e in costante sviluppo URP.
L'applicativo, inoltre, non fa piu` uso di alcun asset non di pubblico dominio. Questo ha di fatto eliminato le problematiche dovute al supporto di sviluppatori terzi verso gli aggiornamenti del motore grafico e verso l'ottenimento di valori e/o parametri non esposti attraverso la documentazione di questi asset.

La resa grafica della scena può dichiararsi complessivamente migliorata.
La presenza di un fiume dal corso più irregolare e una maggiore attenzione della modellazione della scena rendono i video maggiormente simili a quelli ottenibili tramite misurazioni sul campo.
Il lavoro effettuato sulla nuova scena ha portato all'eliminazione di numerosi artefatti grafici dovuti alla reazione delle luci con il vecchio shader dell'acqua derivante dal pacchetto di assets "Cascade".
Tra i "glitch" rimossi ricordiamo in particolare quello che causava il contrasto elevato delle foglie in casi di bassa illuminazione e l'effetto di "stacco" netto che era presente tra l'acqua e la riva del fiume dovuto ai settaggi di rifrazione e riflessione troppo accentuati.
La risoluzione di questi problemi ha mostrato come, nel caso di scene con illuminazione ridotta, l'algoritmo di OTV si trovasse in difficoltà a tracciare correttamente gli oggetti in movimento sul corso del fiume, segnalando di fatto agli sviluppatori una criticità del loro algoritmo.
Nota a termine, nello sviluppo del progetto non si misurata alcuna degradazione delle performance dell'applicativo rispetto alla versione precedente.

Sono riportati due frame tratti dalle diverse versioni dell'applicativo.
In alto l'immagine tratta dalla prima versione e in basso quella tratta dalla seconda.
Sono evidenti la rimozione degli artefatti grafici di luce ed ombre sulle sponde del fiume e la migliore resa grafica complessiva della nuova scena.

I nuovi strumenti messi a disposizione dall'applicativo semplificano notevolmente il lavoro dell'utilizzatore in fase della generazione dei video.
Lo stesso Riccardo, che ha utilizzato per qualche giorno il software, mi ha segnalato come grazie alla nuova interfaccia grafica sia diventato molto più semplice configurare l'applicazione per esportare video sintetici utili.

In conclusione, le misurazioni effettuate sembrano confermare ulteriormente i progressi fatti.
L'output dell'algoritmo di OTV si è dimostrato meno dipendente dai parametri inseriti, inoltre i suoi risultati hanno errori molto bassi rispetto ai valori di "ground-truth" in condizioni ottimali.
In condizioni di illuminazione particolari, come in presenza di molta luce riflessa o scarsa illuminazione, l'algoritmo di OTV mostra risultati meno precisi, segnale che probabilmente è necessario ulteriore lavoro per adattarlo a queste casistiche.

5.2 Sviluppi futuri

Il programma RFSB puo` considerarsi in uno stato maturo, tuttavia rimangono alcune funzionalità migliorabili per aumentare la qualità dei video sintetici.

L'applicativo attualmente ha due soli tipi di oggetti traccianti che vengono generati sul corso d'acqua.
Per aggiungere variabilità alle scene generate si potrebbe procedere creando un gruppo di oggetti traccianti molto più ampio e variabile nel colore e nelle forme.
Ad esempio, sarebbe utile affiancare alle foglie d'albero anche rametti e rifiuti.
La creazione di questa variabilità degli oggetti generati potrebbe rendere le scena generate maggiormente robuste in caso di utilizzo di algoritmi di OTV basati su machine learning.
Per procedere in questo senso, nella progetto sono già importate circa una cinquantina di texture di foglie diverse, è dunque necessario solo creare i corrispettivi materiali e prefabs da referenziare all'emettitore del fiume.

Durante i test, il laureando Riccardo Turra ha proposto di inserire all'interno del benchmark degli elementi di interferenza che possano andare ad essere scambiati dal benchmark con traccianti.
Un esempio di interferenza potrebbe essere un uccello che vola nel campo visivo della telecamera andando di fatto ad essere scambiato dagli algoritmi di OTV per tracciante.
Purtroppo, non ho avuto tempo per ricercare ed implementare questa funzionalità,

tuttavia penso potrebbe essere interessante misurare come gli algoritmi rispondono anche a interferenze presenti nei video.

Sia la versione corrente che quella precedente del benchmark permettevano la generazione di video con camere stereo.
I video generati non sono ancora stati utilizzati per alcun test in quanto richiedono del post-processing tramite programmi esterni per dividere l'output delle due camere stereo in file video separati.

È riportato un frame della nuova versione dell'applicativo in cui è stato abilitato il rendering tramite le camere stereo.

L'analisi delle immagini sintetiche stereo potrebbe mettere in risalto eventuali problematiche utili al miglioramento degli algoritmi testati.

Bibliografia

[1] F. Tauro, F. Tosi, S. Mattoccia, E. Toth, R. Piscopia, S. Grimaldi, "Optical Tracking Velocimetry (OTV): leveraging optical flow and trajectory-based filtering for surface streamflow observations", Remote Sensing, 2018, 10(12), 2010

[2] Simone Dosi, “Studio di metodologie per generazione di dataset sintetici”, Attività di tirocinio, Università di Bologna, AA 2018/19

[3] Riccardo Turra, “Test su algoritmi di Optical Vision”, Tesi di Laurea, Università di Bologna, AA 2018/19

[4] Sebastian Lague, “Curve Editor”, github.com/SebLague/Path-Creator

[5] Wikipedia.org, Curva di Bézier, wikipedia.org/wiki/Curva_di_Bézier

[6] Wikipedia.org, Gaussian Blur, wikipedia.org/wiki/Gaussian_blur

[7] Alex Vlachos, “”Water flow in Portal 2”, Valve, 28/7/2010, SIGGRAPH, steamcdn-a.akamaihd.net/apps/valve/2010/siggraph2010_vlachos_waterflow.pdf

[8] Wikipedia.org, Normal mapping, wikipedia.org/wiki/Normal_mapping

[9] Inigo Quilez, Texture Repetition, iquilezles.org/www/articles/texturerepetition/texturerepetition.htm

[10] Unity.com, Prefabs, docs.unity3d.com/Manual/Prefabs.html

[11] Unity.com, User interfaces (UI), docs.unity3d.com/Manual/UIToolkits.html

[12] Unity.com, Scriptable Object, docs.unity3d.com/Manual/class-ScriptableObject.html

[13] Unity.com, CreateAssetMenuAttribute, docs.unity3d.com/ScriptReference/CreateAssetMenuAttribute.html