Progetto di Interactive 3D Graphics di Andreussi Francesco (matricola 118708)

Il mio progetto consiste nella realizzazione di una versione virtuale (e tridimensionale) del noto gioco da tavolo “Forza 4!”.
Il supporto verticale e le pedine che servono per giocare hanno come materiale uno `ShaderMaterial` che gli assegna un “effetto plastica” grazie alla composizione di un microfacet e di un lambert shader (posta nel fragment shader). La composizioneavviene attraverso l'operatore `mix` di `glsl` (interpolazione lineare tra due valori) che prende come valore interpolante una uniform che varia da materiale a materiale. Infine ho aggiunto (mediante una somma) un colore ambientale per rendere la zona non illuminata meno scura e il materiale più realistico, inoltre avere la parte in ombra di colore nero rendeva il gioco difficilmente utilizzabile poiché si distingueva difficilmente la differenza tra pedine rosse e gialle durante il gioco.
Questi elementi poggiano su un tavolo rotondo che, nelle mie intenzioni, dovrebbe avere una superficie in legno trattato in modo da essere molto lucido, questo materiale è stato realizzato in maniera analoga al primo, ma con un diverso fragment shader, in grado di gestire le texture. Il piedistallo dello stesso, invece e realizzato con un colore grigio/nero uniforme con una prevalenza di componente lambert rispetto a quella microfacet.
Accanto al tavolo si trovano due sedie, il cui materiale è ispirato alle sedie presenti in casa mia, la cui superficie sarebbe di legno, ma è totalmente coperta da una laccatura semi-lucida di colore grigio/nero. Il materiale delle sedie è stato ripreso per il mobile basso e la cornice addossati alla parete di fronte alla finestra, il quale è dello stesso materiale della porta: un legno scuro, meno lucido rispetto al tavolo realizzato con lo stesso shader, ma con valori e texture diverse.
Porta e finestra condividono anche il materiale delle proprie maniglie: materiale realizzato con gli stessi shader degli altri materiali con colore uniforme, ma in questo caso non è presente la componente lambertiana, infatti l'effetto voluto è quello di un materiale metallico che ricordi l'ottone.
All'interno della cornice è presente un famoso dipinto di Piet Mondrian *“Composizione N° ll in rosso, blu, nero e giallo” (1929)* , che riflette la luce seguendo il modello lambertiano (lo shader è lo stesso delle altre superfici texturizzate, ma il mix elimina la componente speculare).
Ovviamente sono puramente lambertiani sono anche i muri, il soffitto, il tetto, e la parte di casa sotto il pavimento. Questi elementi hanno un ambient color grigio medio e tutti, tranne il tetto a cui è applicata una texture (avrei voluto applicargli anche normal e displacement map, ma non sono riuscito ad ottenere dei risultati soddisfacenti), sono totalmente bianchi.
Il pavimento, invece, ha un effetto legno trattato, ma in maniera più lieve rispetto a quello del tavolo; infatti in questo caso si nota la rugosità (ottenuta tramite l'applicazione di una normal map) del materiale che si presenta comunque semi-lucido.
L'ambiente esterno è costituito, in maniera molto grossolana da uno skybox costruito da `three.js` e da un piano dove appoggia la casa; lo skybox usa uno shader predefinito che non lo fa interagire con la luce mentre il piano, a cui ho applicato una texture, si comporta come un materiale lambertiano.
Tutte le texture sono applicate in maniera ripetuta alle superfici, infatti nello shader, nel comando `texture2D` che prende in input una texture e una coppia di coordinate uv (una coppia di float compresi tra 0 e 1) modifico le coordinate uv passate moltiplicandole prima per una potenza di due passata come `uniform` e poi passando a `texture2D` il resto della divisione tra il risultato della moltiplicazione e 1, cosicché, al variare della `uniform`, si possa modificare il numero di volte che la texture viene ripetuta.
Nel progetto ho inserito tre luci: una direzionale, cioè dove la potenza della luce non decade col crescere della distanza dalla stessa; questa è posta in direzione, normalizzata, `(0.25,0.5,0.25)`. All'interno della casa ci sono due sorgenti luminose: una plafoniera al

centro del soffitto e una lampada a terra alla sinistra di quadro e mobile.
I vetri delle lampade hanno una forte componente ambientale che li rende molto luminosi e il loro materiale è ottenuto sempre attraverso i soliti shader, ma in questo caso la componente lambertiana è nulla e l'alfa molto elevata (200) fa in modo che siano talmente lucidi che sembra addirittura si lascino attraversare dalla luce (non credo sia la soluzione migliore, però è semplice e abbastanza efficace).
La lampada a terra ha uno stelo metallico di colore scuro.
Le animazioni sono piuttosto semplici e si dividono in due tipologie: transazioni (implementate appoggiandosi alle funzioni della libreria `tween.js`) e rotazioni (gestite tramite quaternioni).
Ho usato le transazioni in maniera piuttosto diffusa; fin da subito, infatti, l'inquadratura parte dal punto `(0,0,0)` per posizionarsi in un punto random all'interno di un cubo 1000 ×600×1000 centrato nell'origine (sempre all'interno della casa) tramite un tweening con easing quadratico con velocità decrescente (`quadratic.out`), terminata l'animazione si può premere il comando `play` che porta, sempre con un tweening quadratico che rallenta nel tempo, la camera nella posizione del giocatore rosso, a cui si concatena un'animazione più complessa che dapprima, tramite una transazione analoga alle altre porta una pedina sopra il supporto del gioco e poi la gira di 90° per metterla in posizione verticale, mediante l'uso di un quaternione. Per giocare l'utente può selezionare la colonna spostando la pedina a destra e a sinistra (data la brevità del movimento non ho inserito animazioni in questo punto) e poi, quando la lascia cadere la pedina è animata con un tweening con easing di tipo `bounce.out` per dare un effetto fisico di caduta e collisione che però è calcolato "a priori" trovando la prima riga libera della colonna selezionata e ponendo come target le coordinate corrispondenti alla coppia `(riga, colonna)` selezionata. Al cambio turno la camera si sposta sempre con velocità quadratica decrescente fino ad arrivare nella posizione di gioco dell'avversario, le quali sono uguali a meno della z, che è opposta. Durante la traslazione viene effettuata in automatico anche una rotazione di 180° poiché la camera ha un attributo `lookAt=(0,40,0)`, costante.
Premendo `pause` la visuale si sposterà, con gli stessi soliti parametri, in una posizione qualsiasi del cubo sopra menzionato, mentre lo `startNewGame` oltre ad eseguire tale spostamento, resetta le strutture dati di supporto e la configurazione delle pedine del gioco, in modo che sia pronto per un'altra partita.
Per il post-processing ho inserito in alto al centro un checkbox (come è stato fatto negli esempi di `three.js` e nelle risorse del corso) che abilita/disabilita un effetto negativo, dove il si mostra a video il colore contrario a quello normalmente utilizzato, l'effetto è volutamente semplice cosicché da mantenere il gioco a 60 fps.
Infine, il vincitore potrà apprezzare una scritta dello stesso colore e materiale delle proprie pedine che annuncia la sua vittoria e un "effetto coriandoli" ottenuto grazie ad un sistema particellare piuttosto semplice.
Avrei voluto aggiungere anche le ombre, ma, a quanto pare, la versione `r74` di `three.js` non le supporta (nel file `three.min.js` manca il comando `THREE.ShaderChunk("shadowmap_fragment")` , però ho fatto in modo che le luci interne non influiscano sull'ambiente esterno alla casa e che la luce ambientale non illumini le facce interne dei muri o il pavimento o i mobili, questo grazie ad un controllo che se il frammento preso in considerazione ha una `worldPosition` tale da essere all'esterno della casa (`|x|>=1390` oppure `y>=870` oppure `|z|>=1190`) questo riceve solo la luce direzionale posta all'esterno della casa, altrimenti la luce direzionale non influisce sul suo colore finale sul quale, però, influiscono le due luci poste internamente alla casa.

P.S.: il progetto è stato testato usando Safari su un MacBookPro con OS X El Capitan e scheda grafica Intel HD 4000 e mantiene sempre i 60 fps.