Universitá degli Studi di Ferrara

Corso di Grafica Computerizzata

Progetto Finale:
Car Driving

Professore
Di Domenico
Giovanni

Studente
Ferro Enrico
Matricola 11214

Anno Accademico 2016/2017

Introduzione

Nell'applicazione grafica presentata di seguito è stato implementato un piccolo *car driving*, ovvero una sorta di piccolo simulatore che permette di guidare un'auto attraverso un circuito cittadino chiuso.
L'utente che utilizza l'applicazione ha la possibilità di guidare l'auto, utilizzando la tastiera, attraverso il circuito proposto e di osservare il movimento di quest'ultima all'interno della scena da punti di vista diversi, grazie alla possibilità di poter cambiare la posizione della camera.
Il codice dell'applicazione è distribuito su 4 file: uno in formato HTML (.html), che è il file che si deve mandare in esecuzione qualora si intenda utilizzare l'applicazione, e 3 in formato JavaScript (.js) che servono per creare e modellare la scena. In aggiunta a questi, sono stati utilizzati file libreria, file che implementano modelli tridimensionali e immagini utilizzate come texture; nel proseguo della relazione, alcuni di questi, i più importanti e significativi, verranno presi in esame in maniera tale da poter capire in quale maniera vengano sfruttati dall'applicazione.
Nota importante riguarda il fatto che nell'applicazione non è stata implementata nessun tipo di legge fisica ed in particolare ciò si traduce con il fatto che non vi è alcun tipo di interazione tra l'auto e gli oggetti che la circondano, perciò ogni qualvolta il veicolo viene a "contatto" con un oggetto della scena, sia esso un palazzo, piuttosto che un albero, si limiterà ad oltrepassarlo e non vi sarà alcun tipo di urto.

Il file HTML: progetto_FerroEnrico.html

Il file HTML, come già anticipato, è quello che si deve eseguire per poter utilizzare l'applicazione grafica all'interno di un browser web.
Oltre ad essere il "punto di esecuzione" dell'applicazione, il file rappresenta anche il raccordo tra tutti i file e le librerie utilizzati per creare il *car driving*. Tramite questo file, infatti, vengono caricate tutte le librerie e tutti i file che servono per la corretta renderizzazione della scena.
Infine il file HTML, delimita anche quanto spazio la canvas occuperà all'interno della pagina del browser (in questo caso l'impostazione è *fullscreen*) e fornisce all'utente un'indicazione sui tasti da utilizzare per pilotare l'automobile e per cambiare la camera, informazioni visibili in basso a sinistra qualora si avvii l'applicazione.

Il file JavaScript: progetto_FerroEnrico.js

Il nucleo dell'applicazione è costituito da questo file JavaScript. In esso vengono infatti effettuate le operazioni principali che permettono la renderizzazione e l'animazione della scena. In questo file inoltre viene importato e gestito il modello dell'auto e viene importata la mappa. In realtà la creazione della mappa e degli oggetti contenuti in essa (palazzi, alberi, lampioni, ...) potevano essere tranquillamente inseriti in questo file, ma avrebbero reso il codice estremamente lungo e intricato, oltre che poco modulare. Separando invece il tutto, oltre a rendere il codice più contenuto e più leggibile, qualora si disponga di un'altra mappa sarebbe semplice sostituire la presente, cambiando una sola funzione.
Di seguito vengono presentate le funzioni principali svolte dal file.

Le operazioni preliminari

Le operazioni preliminari e fondamentali per il corretto funzionamento dell'applicazione, visibili nel codice proprio grazie al commento *PRELIMINARY OPERATIONS*, rappresentano la serie di istruzioni che occorrono per creare la scena stessa, la camera, e la finestra di renderizzazione.
Di questa parte di codice fa parte anche la creazione dello sfondo, ottenuto tramite uno skybox e delle luci, una ambientale di un grigio abbastanza chiaro e una direzionale posta volutamente "in fondo" alla scena in maniera da simulare che la luce provenga dal sole che si può intravedere nello skybox.
Sempre inclusa tra le operazioni preliminari vi è l'inclusione della mappa attraverso la funzione *map(scene);* che non fa altro che richiamare il file *map.js* che verrà analizzato in seguito.
Infine, tramite la variabile *stats*, viene eseguita la funzione *initStats()* che permette di avere una statistica sul comportamento dei fotogrammi per secondo (fps), visibile in alto a sinistra.

L'automobile

L'automobile scelta per l'applicazione grafica è una Bugatti Veyron, disponibile tra i modelli messi a disposizione dalla libreria THREE.js (il modello si trova in examples/obj/veyron all'interno della cartella principale three.js-master, disponibile nel sito dedicato alla libreria grafica). Il modello viene caricato nella scena attraverso la funzione *loadPartsBinary()* (funzione resa disponibile importando il file *BinaryLoader.js*) e viene gestito all'interno di essa grazie alle funzioni offerte dal file *Car.js*, sempre presente nella libreria grafica THREE.js. Tramite quest'ultimo file possono essere gestiti molti aspetti del modello che viene caricato, come: la scalatura, il posizionamento delle ruote posteriori, la velocità massima e l'accelerazione dell'auto e tutte quella serie di funzioni, come ad esempio l'effetto "sterzatura" delle ruote anteriori, che servono

per rendere il più realistico possibile il movimento del modello di auto caricato.
La funzione che però aggiunge l'oggetto alla scena (nel codice l'auto viene identificata dalla variabile *veyron*), è *addCar(object, x, y, z, s)*; passando a questa funzione l'oggetto interessato e le coordinate di dove si intende posizionarlo, questa provvede ad aggiungerlo alla scena e ad impostare la camera come sua "inseguitrice".

L'interazione con l'utente

L'interazione con l'utente, in questa applicazione grafica, non è particolarmente articolata: egli può infatti guidare l'auto attraverso il circuito proposto e cambiare la camera di visualizzazione.
La guida, gestita dalle funzioni *onKeyDown(event)* e *onKeyUp(event)*, è piuttosto semplice: ogni qualvolta uno dei tasti viene premuto o rilasciato si scatena un evento che viene "comunicato" alla funzione *render()* la quale provvede a ridisegnare la scena fornendo l'effetto di spostamento dell'auto all'interno di essa.
Per quanto riguarda l'interazione sul cambio di camera, di per sè non vi è nulla di complicato: se l'utente preme uno dei tasti designati alla gestione della camera, essa viene spostata nella posizione associata al determinato tasto; ciò che magari potrebbe essere un po' meno comprensibile è invece come la camera venga gestita dal sistema (nella sottosezione seguente si cercherà di fare un po' di chiarezza su tutto questo). L'utente inoltre, può gestire la camera anche attraverso l'utilizzo del mouse e ciò è reso possibile grazie all'istruzione *var cameraControls = new THREE.OrbitControls(camera, renderer.domElement);*.

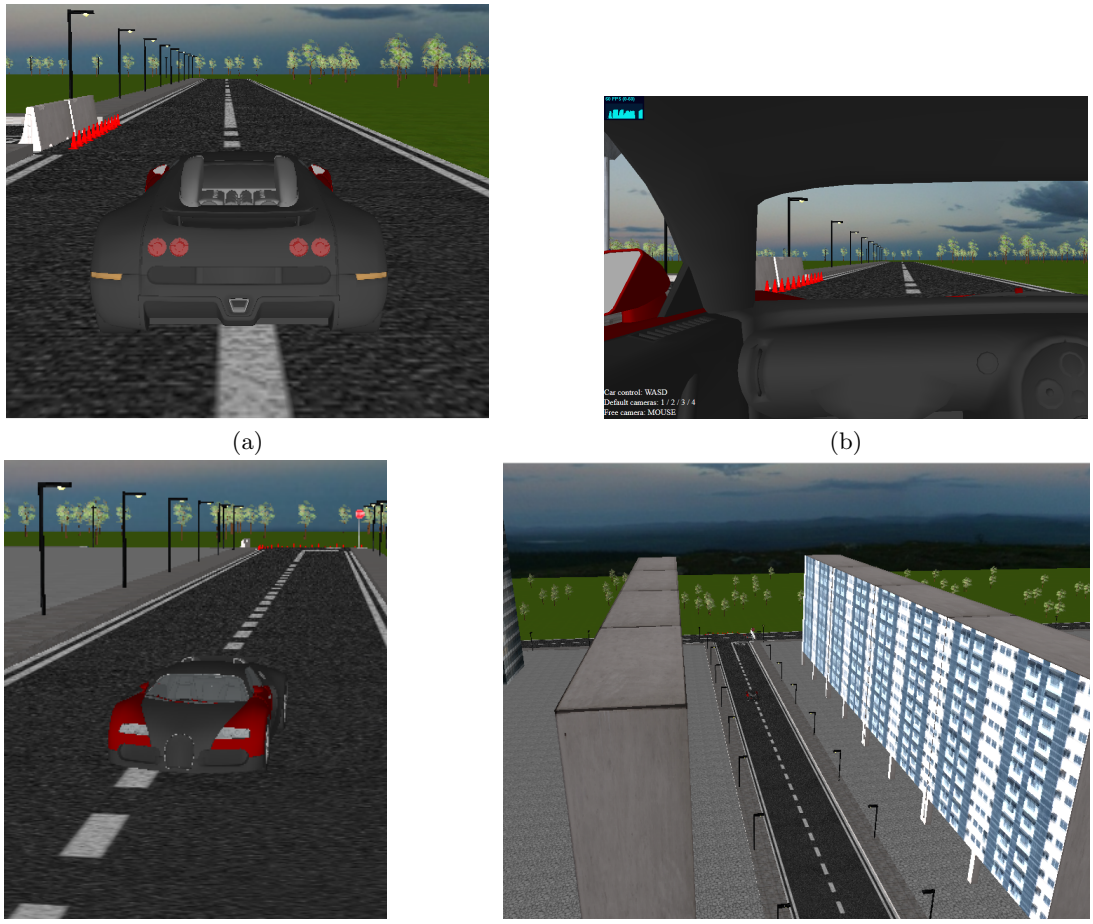
La gestione delle camere di default

A parte la possibilità di gestire liberamente la camera attraverso l'uso del mouse, l'utente può decidere di cambiare la visualizzazione della scena scegliendo una tra le 4 camere di default che l'applicazione offre: 3 di esse sono ad "inseguimento" (selezionabili con i tasti 1/2/3), ciò significa che si spostano qualora l'auto si stia muovendo, mentre una è fissa e "segue" l'auto soltanto ruotando sul proprio asse y (si può selezionare con il tasto 4).
Fondamentale per distinguere le due tipologie di camere è la variabile globale *FOLLOW_CAMERA*. Tale variabile, di tipo booleano, se vera, nella funzione *addCar(object, x, y, z, s)*, fa in modo che la camera segua per l'appunto il veicolo. La differenza tra le tre camere ad inseguimento sta nella posizione in cui questa è posta rispetto al veicolo: la prima, quella con cui la scena viene caricata, dietro al veicolo, la seconda al suo interno e l'ultima di fronte (Figura 1 (a), (b), (c)).
La quarta camera di default disponibile, invece, come già anticipato, non segue "fisicamente" l'automobile, ma rimane fissa al centro della scena e ruota su se stessa (Figura 1 (d)). Con questa inquadratura, non è detto che il veicolo sia sempre visibile in quanto, potrebbe essere nascosto dai palazzi. Quando questa camera è attiva, la variabile booleana *FOLLOW_CAMERA*, viene impostata su *false* e di conseguenza la camera non si sposta "fisicamente" insieme all'auto.
Tutto ciò che riguarda le camere di default, viene gestito dalla funzione *setCurrentCamera(car, cameraType)*, che in base ai valori che le vengono passati qualora si prema uno dei tasti che gestiscono la camera, seleziona l'inquadratura corrispondente.

La renderizzazione

Tutto ciò che è visibile nella scena è dovuto alla funzione *render()*. Quest'ultima provvede a "disegnare" tutti gli oggetti della scena e a creare l'effetto di movimento dell'auto all'interno

di essa. Inoltre, l'aggiornamento delle statistiche relative agli fps, vengono aggiornate ogni qualvolta questa funzione viene eseguita.



(a) (b)

(c) (d)

Figura 1: *Scorci delle varie camere: (a)back (b)internal (c)front (d)center.*

La mappa

Il secondo dei tre file JavaScript che costituiscono l'applicazione, ha il compito di creare la mappa sulla quale poi viene ricavato il circuito su cui l'utente avrà la possibilità di condurre l'auto. La struttura è estremamente semplice: se la si guarda dall'alto si possono notare 3 strade in verticale poste tutte alla stessa distanza e 3 orizzontali, anche queste poste tutte alla medesima distanza. Gli incroci delle sei strade realizzano una sorta di 4 quadrilateri, 3 dei quali presentano al loro interno dei palazzi, mentre nel quarto è stato ricavato un parco con delle piante e con alcuni giochi per bambini.
Nel file in questione viene creato il suolo (strade, marciapiedi, erba, ...) e vengono importati, tramite le apposite funzioni, tutti gli oggetti che su di esso poggiano (palazzi, lampioni, alberi, ...).

Figura 2: *Vista dall'alto della mappa utilizzata nell'applicazione. In giallo sono segnate le avenue mentre in arancione le street. I numeri rappresentano gli angoli mentre le lettere gli incroci.*

Le strade ed i marciapiedi

Nella parte iniziale del file *map.js* vengono caricate tutte le texture, e con esse si creano i materiali, che servono per le varie parti che compongono il suolo.
Dopo questa primna parte vengono create le strade, realizzate con un piano attraverso la funzione *THREE.PlaneGeometry*: esse vengono divise in street ed in avenue (Figura 2). Per raccordare street ed avenue sono utilizzati altri piani ai quali sono applicate texture aventi strade con linee continue piuttosto che con la segnaletica orizzontale classica degli stop. I raccordi

sono di due tipi: *angle*, distinti attraverso un numero, e *cross*, ognuno contraddistinto da una lettera.
Lungo i perimetri delle 4 piazze che si vengono a creare incrociando le 6 strade, sono stati creati dei marciapiedi, dei *THREE.BoxGeometry* ai quale viene applicata una texture che ricorda una pavimentazione. Anche in questo caso, per distinguere i marciapiedi "verticali" da quelli "orizzontali" (posizioni relative alla piantina di fig. 2), si utilizzano i commenti street ed avenue. Risulta infine utile notare che insieme al codice di creazione dei marciapiedi, viene chiamata la funzione *lamp*. Essa inserisce nei marciapiedi i lampioni il cui modello viene creato nel file *objects.js* e verrà discusso nella sezione dedicata a questo.

Le piazze pavimentate ed i terreni erbosi

Come è visibile in figura 2, dove non vi sono le strade o i marciapiedi, il suolo è pavimentato o simula un manto erboso. La realizzazione di questo tipo di suoli è ancora una volta resa possibile creando dei piani ed applicando ad essi le texture che simulano per l'appunto un pavimento di ciottoli ed un prato.
Interessante risulta notare come il caricamento della texture che rappresenta il manto erboso venga ripetuto due volte: infatti nella riga 532 del codice è presente il commento *reload texture*, proprio a testimonianza di questo fatto. Questo è stato necessario per applicare la texture senza valori di *repeat* e di *wrapping* nei piccoli piani che formano confinano con gli angoli della strada (*angle* nel codice). Questa tecnica verrà spesso riutilizzata nel file *objects.js* per dare a forme diverse la stessa texture.
Nota importante da fare è il fatto che, come era stato detto nell'introduzione, non è stata implementata alcuna legge fisica e quindi, qualora l'utente diriga l'auto verso una delle piazzole, che si trovano ad un'altezza leggermente superiore rispetto alle strade ed al parco esterno, l'auto darà come l'impressione di affondare le proprie ruote nel suolo.

L'importazione degli oggetti

L'ultima parte di codice aggiunge alla mappa alcuni oggetti tridimensionali tipici di un ambiente urbano e gli oggetti che servono per "chiudere" determinate vie in maniera tale che l'auto sia vincolata a viaggiare su un circuito chiuso. Per tutti gli oggetti, la procedura di import è analoga: si chiama la funzione corrispondente e ad essa si passano le coordinate in cui si intende posizionare l'oggetto, l'angolo di rotazione che ad esso si vuole imprimere lungo l'asse y, oltre che la variabile *scene*. Meritano un appunto gli alberi che vengono creati nelle aree verdi esterne alle strade, le quali coordinate cambiano ogni qualvolta si riavvii l'applicazione, in quanto generate tramite una funzione random.

Gli oggetti della mappa

Per abbellire la scena e renderla il più possibile simile ad un ambiente cittadino reale, alla mappa, è stato necessario aggiungere una serie di oggetti tridimensionali tipici di questa tipologia di scenario come possono essere lampioni, palazzi ed alberi. Gli oggetti possono essere divisi principalmente in due categorie: oggetti tridimensionali creati attraverso le funzioni geometriche messe a disposizione della libreria THREE.js ed oggetti tridimensionali dalle forme più complesse, creati in formato .obj ed importati nella scena attraverso un'opportuna procedura.
Poichè la procedura di creazione di ogni oggetto appartiene o alla prima o alla seconda categoria, di seguito verranno descritti soltanto i passaggi chiave di ognuna delle due tipologie, in maniera tale da non rendere tale relazione una continua ripetizione di concetti.

Gli oggetti creati sfruttando la libreria THREE.js

A questa categoria appartengono:

- tutti i palazzi
- i lampioni
- i segnali di stop

Come già accennato non verranno discussi gli oggetti uno per uno, ma verrà fornita una panoramica generale su come questa categoria di oggetti è stata creata.
Ad esclusione del *palace2*, formato semplicemente con un *THREE.BoxGeometry* al quale vengono applicate un paio di texture, il resto degli oggetti sfrutta la cosiddetta relazione parent-child messa a disposizione dalla libreria grafica; ciò ne consegue che ad esclusione di uno, tutti gli altri oggetti facenti parte di questa "categoria", non sono altro che una serie di forme tridimensionali aggregate insieme in un *THREE.Object3D()*. Questa funzione offre la possibilità di mettere insieme oggetti aventi forme, dimensioni e materiali diversi in maniera tale da creare un unico oggetto come ad esempio un segnale stradale.
Altra tecnica usata per questi oggetti è la *THREE.MultiMaterial(materials)*, istruzione che permette di applicare ad un solido materiali diversi per le diverse facce che lo compongono. Passando a questa istruzione un vettore contenente vari tipi di materiali, la funzione andrà a posizionarli nelle varie facce del solido, così facendo, per esempio è stato possibile applicare la texture che rappresenta le finestre di un palazzo, nelle facce laterali di un parallelepipedo e la texture che rappresenta il cemento nella faccia superiore.
Infine, c'è da sottolineare che, come accennato precedentemente, per alcuni oggetti la stessa texture è stata caricata più volte in maniera tale da ottenere effetti di *wrapping* differenti.

Gli oggetti in formato .obj

Gli oggetti dalle forme più complesse, sono invece realizzati tramite apposite applicazioni che permettono poi il salvataggio nel formato object (.obj), facilmente importabile nelle applicazioni grafiche implementate con THREE.js attraverso le funzioni messe a disposizione dal file *OBJLoader.js*.
In questa applicazione gli oggetti importati sono:

- i coni stradali
- le barriere New Jersey
- gli alberi
- i giochi per bambini (scivolo, cavalluccio, giostra rotante)

Fulcro dell'importazione degli oggetti tridimensionali all'interno dell'applicazione è la funzione *THREE.OBJLoader()*. Questa funzione importa dal percorso dichiarato dal programmatore l'oggetto e lo rende disponibile; a quest'ultimo può essere poi associato un determinato materiale e su di esso si possono effettuare ingrandimenti e riduzioni attraverso l'istruzione *object.scale.set(x,y,z)*. In realtà, nel codice, alla funzione *THREE.OBJLoader()* viene passata una variabile denominata *manager*: questa variabile, di tipo *THREE.LoadingManager()* ha funzione di debug, infatti essa viene associata ad una funzione che opera nella console.
Aspetto interessante è il fatto che, poichè a questi oggetti è applicabile un qualsiasi materiale, si può abbinare a loro anche una texture ed è proprio combinando questa possibilità con l'opzione *child.material.transparent = true* che si è riusciti ad ottenere un discreto realismo per le foglie degli alberi.
Anche in questo caso, come per gli oggetti creati con le funzioni geometriche messe a disposizione da THREE.js, l'utilizzo di questi elementi tridimensionali risulta estremamente semplice: basta chiamare la funzione associata all'elemento che si intende aggiungere, passando ad essa le 3 coordinate spaziali, l'angolo di rotazione sull'asse verticale e la variabile *scene* e il gioco è fatto.