

ALMA MATER STUDIORUM · UNIVERSITÀ DI BOLOGNA

SCUOLA DI INGEGNERIA E ARCHITETTURA
DIPARTIMENTO DI INFORMATICA - SCIENZA E INGEGNERIA

CORSO DI LAUREA MAGISTRALE IN
INGEGNERIA INFORMATICA

ANALISI DI WEBGI PER LO SVILUPPO E L'OTTIMIZZAZIONE DI SCENE VIRTUALI 3D

TESI DI LAUREA IN
FONDAMENTI DI COMPUTER GRAPHICS M

RELATORE:

**Prof.
GIULIO CACCIOLA**

CANDIDATO:

GRECO DAVIDE

CORRELATORE:

**Dr.
ALESSANDRO PUCCINI**

MATR. NR.: 0001001580

SESSIONE III

ANNO ACCADEMICO 2021/2022

Sommario

Indice

| | |
|---|------|
| Elenco delle figure | VI |
| Elenco delle tabelle | VIII |
| Elenco dei Codici | IX |
| Acronimi | X |
| 1 Stato dell'arte | 1 |
| 1.1 Il mondo del web e della grafica 3D | 1 |
| 1.1.1 HTML5 | 1 |
| 1.1.2 WebGL | 2 |
| 1.2 Pipeline di rendering | 3 |
| 1.3 Three.js | 8 |
| 1.4 I principali concetti di modellazione 3D | 10 |
| 1.5 Attività di tirocinio ed obiettivi | 12 |
| 1.6 Analisi del software | 13 |
| 1.6.1 Creazione del modello 3D | 13 |
| 1.6.2 Funzionamento del software | 14 |
| 1.6.3 Ricerca di metodi e strumenti per migliorare il servizio aziendale . | 15 |
| 2 WebGi SDK | 19 |
| 2.1 Strumenti per l'uso di WebGi | 20 |

| | | |
|----------------------------------|--|----|
| 2.1.1 | Node.js | 20 |
| 2.1.2 | Editor di codice | 20 |
| 2.1.3 | GitHub | 20 |
| 2.1.4 | Parcel | 21 |
| 2.2 | La struttura di WebGi | 21 |
| 2.2.1 | 3D Viewer Core | 22 |
| 2.2.2 | Asset Management e importazione | 28 |
| 2.2.3 | Materiali e shaders | 29 |
| 2.2.4 | Screen Space Post Processing | 34 |
| 2.2.5 | Luci e setup della scena | 40 |
| 2.2.6 | Animazioni | 41 |
| 2.2.7 | Export e render | 43 |
| 2.2.8 | Plugin di supporto | 45 |
| 2.2.9 | Plugin e ShaderMaterial personalizzati | 47 |
| 2.2.10 | Esempio di creazione di una scena | 50 |
| 2.2.11 | Feature in sviluppo e limiti | 53 |
| 3 | Sviluppo del Progetto | 55 |
| 3.1 | Analisi dei requisiti | 55 |
| 3.2 | Analisi del problema | 56 |
| 3.3 | Sviluppo del progetto | 58 |
| 4 | Conclusioni | 71 |
| 4.1 | Limiti | 72 |
| 4.2 | Sviluppi futuri | 72 |
| Riferimenti bibliografici | | 73 |

Elenco delle figure

| | | |
|------|---|----|
| 1.1 | Architettura della pipeline di rendering. | 4 |
| 1.2 | Visualizzazione di una mesh 3D composta da triangoli con diversi livelli di dettaglio. | 11 |
| 1.3 | Materiali che è possibile creare in Blender, combinando texture e le proprietà del materiale. | 12 |
| 2.1 | Orientamento dei raggi luminosi in base alla superficie di incidenza. | 29 |
| 2.2 | Noise Bump Mapping applicato ad una mesh sferica. | 33 |
| 2.3 | Comparazione dell'effetto visivo fra Parallax e Bump mapping. | 33 |
| 2.4 | Progressive Plugin disattivato vs attivato in WebGi. | 35 |
| 2.5 | Bloom Plugin disattivato vs attivato in WebGi. | 36 |
| 2.6 | SSAO Plugin disattivato vs attivato in WebGi. | 37 |
| 2.7 | SSR Plugin disattivato vs attivato in WebGi. | 39 |
| 2.8 | Ground Plugin disattivato vs attivato in WebGi. | 41 |
| 2.9 | Editor online in WebGi. | 44 |
| 2.10 | Material configurator nell'editor online di WebGi. | 46 |
| 2.11 | Scena prodotta del tamplate di WebGi. | 52 |
| 2.12 | Funzionamento del ray-tracing. | 53 |
| 3.1 | Visualizzazione del mobile intero in WebGi. | 59 |
| 3.2 | Resa grafica del SSAO e del Bloom sul mobile. | 60 |
| 3.3 | Visualizzazione dell'SSR sul mobile. | 61 |

| | | |
|-----|---|----|
| 3.4 | Materiali PBR con parallax mapping e reflections. | 62 |
| 3.5 | Cambio del materiale attraverso i bottoni dell'infografica. | 67 |
| 3.6 | Schermata di caricamento della scena. | 70 |

Elenco delle tabelle

Elenco dei Codici

| | | |
|------|--|----|
| 1.1 | Esempio di come utilizzare un canvas in HTML5 | 2 |
| 1.2 | File HTML della scena presa come esempio | 6 |
| 1.3 | File JS della scena presa come esempio | 6 |
| 1.4 | Funzione per disegnare gli oggetti sul canvas nella scena d'esempio. | 8 |
| 1.5 | Creazione di una semplice scena 3D con Three.js | 10 |
| 2.1 | Creazie del viewer attraverso l'elemento canvas | 26 |
| 2.2 | Creazie del viewer attraverso il container | 26 |
| 2.3 | Creazie del viewer attraverso il canvas con opzioni per il rendering | 26 |
| 2.4 | Esempio base di utilizzo dell'asset manager plugin in WebGi. | 28 |
| 2.5 | Esempio base di utilizzo del GLTF Animation Plugin. | 42 |
| 2.6 | Esempio base di utilizzo del Picking Plugin. | 45 |
| 2.7 | Sviluppo di una UI per la configurazione dei materiali. | 46 |
| 2.8 | Interfaccia IViewerPlugin in WebGi. | 47 |
| 2.9 | Esempio base di plugin personalizzato fornito da WebGi. | 48 |
| 2.10 | File .html presente nel template di WebGi. | 50 |
| 2.11 | File index.ts presente nel template di WebGi. | 51 |
| 3.1 | File index.ts presente nel template di WebGi. | 58 |
| 3.2 | Funzione per il picking degli oggetti. | 62 |
| 3.3 | Funzione per il focus e l'unfocus degli oggetti. | 63 |
| 3.4 | Funzione per rendere visibile l'infografica. | 64 |
| 3.5 | Gestione delle animazioni attraverso i bottoni. | 65 |
| 3.6 | Funzione per la creazione e gestione del pannello di cambio dei materiali nell UI. | 67 |
| 3.7 | Funzioni per la realizzazione della schermata di caricamento. | 68 |

Acronimi

Capitolo 1

Stato dell'arte

All'interno del capitolo viene fornita una panoramica informativa del mondo 3D e dei suoi strumenti per realizzare scene virtuali 3D in real time, ponendo l'attenzione sulla parte di modellazione, ma soprattutto su quella che è la pipeline di sviluppo di una scena, cercando di capire chi e come entrano in gioco i vari strumenti.

1.1 Il mondo del web e della grafica 3D

Il mondo del web 3D si riferisce all'utilizzo di tecnologie 3D per creare contenuti interattivi per il web. Per poter addentrarsi nel dettaglio del mondo del web 3D, è importante parlare di HTML5 e di WebGL.

1.1.1 HTML5

HTML5 è un linguaggio di marcatura utilizzato per strutturare e descrivere il contenuto di una pagina web. E' la quinta versione del linguaggio HTML (Hypertext Markup Language) e rappresenta una significativa evoluzione rispetto alle versioni precedenti.

HTML5 introduce una serie di nuovi elementi e attributi che consentono di creare contenuti video, audio, mappe, moduli di input e altro, senza la necessità di plugin esterni. Questo significa che i contenuti possono essere visualizzati e utilizzati direttamente all'interno del browser web.

Inoltre, HTML5 include nuove funzionalità come l'API di geo-localizzazione, l'API di canvas per il disegno 2D e 3D, l'API di drag-and-drop e l'API di Web Storage per la gestione dei dati.

Proprio l'API di canvas risulta essere un elemento centrale nello sviluppo di scene virtuali per i web.

L'elemento *canvas* in HTML5 è la porzione di una pagina web in cui è possibile disegnare e manipolare immagini utilizzando script scritto in JavaScript. Il *canvas* è quindi una zona di disegno bidimensionale o tridimensionale in cui è possibile creare immagini dinamiche e interattive, come giochi, grafici, animazioni e mappe.

Il contesto del disegno può essere 2D o 3D (con WebGL) e offre una serie di funzioni per disegnare linee, forme, testo e immagini.

```

1 <canvas id="redRect" width="500" height="300"></canvas>
2 <script>
3   var canvas = document.getElementById("redRect");
4   var ctx = canvas.getContext("2d");
5   ctx.fillStyle = "red";
6   ctx.fillRect(0, 0, 150, 75);
7 </script>
```

Listing 1.1: Esempio di come utilizzare un canvas in HTML5

In questo esempio, viene creato un canvas di 500x300 pixel con id "redRect". Successivamente un contesto 2D viene ottenuto tramite `canvas.getContext("2d")` e quindi utilizzato per disegnare un rettangolo colorato di rosso.

1.1.2 WebGL

Una delle librerie più famose per la realizzazione di grafica 3D per il web è WebGL (Web-Based Graphics Library): è una libreria JavaScript che consente di utilizzare le funzionalità di rendering 3D del browser per creare contenuti interattivi.

È uno standard aperto basato su OpenGL ES, una versione ridotta di OpenGL: libreria per il rendering 3D utilizzata comunemente nei software di progettazione 3D e per sviluppare giochi.

WebGL utilizza il contesto WebGL del browser per accedere alle funzionalità di rendering 3D del computer dell'utente. Una volta che il contesto WebGL è stato creato, è possibile

utilizzare le funzioni di WebGL per creare oggetti 3D, applicare texture e materiali, e utilizzare le funzionalità di rendering avanzate come il rendering di ombre.

WebGL utilizza una serie di chiamate API per creare e modificare gli oggetti 3D, è supportato da tutti i principali browser, inclusi Chrome, Firefox, Safari, Edge e Opera. Ciò significa che i contenuti 3D creati con WebGL possono essere visualizzati su qualsiasi dispositivo che supporta uno di questi browser, quindi computer, tablet e smartphone.

Questa tecnologia per la creazione di contenuti 3D per il web, può essere complessa da utilizzare per gli sviluppatori che non hanno esperienza con la programmazione 3D.

Ci sono diverse librerie JavaScript come Three.js che forniscono un’interfaccia semplificata per lavorare con WebGL e che possono essere utilizzate per aiutare il processo di creazione di contenuti.

1.2 Pipeline di rendering

La pipeline di rendering è un insieme di processi che consiste nel trasformare la scena 3D, data una certa posizione e orientamento della camera, in un’immagine 2D che rappresenta la scena 3D dal punto di vista della camera.

Ad alto livello, la pipeline di rendering viene realizzata dalla GPU (Graphics Processing Unit), che, dopo aver ricevuto i vertici da elaborare e disegnare dalla CPU esegue un un’insieme di operazioni atte a disegnare sullo schermo i pixel dell’immagine. Dalla figura si nota che la CPU è collegata ad un bus che comunica direttamente con la scheda grafica: trasmette i vertici da convertire e poi disegnare nel sistema di coordinate 3D. Inizialmente, è necessario definire uno shader: è un programma che viene eseguito sulla CPU che elabora e manipola dati grafici durante la pipeline di rendering. Il vertex shader è incaricato di prendere in input le coordinate 3D dei vertici dei triangoli per convertirli in vertici in coordinate 2D applicando trasformazioni come rotazioni, traslazioni e scalature. Va specificato che i vertex shader operano sempre su un singolo vertice di input e producono un singolo vertice di output.

Una volta ottenuti i vertici in coordinate schermo entra in gioco il rasterizzatore, il suo compito è quello di trasformare i triangoli della geometria in pixel colorati. Durante la fase di rasterizzazione, il rasterizzatore suddivide ogni triangolo della geometria in una serie di

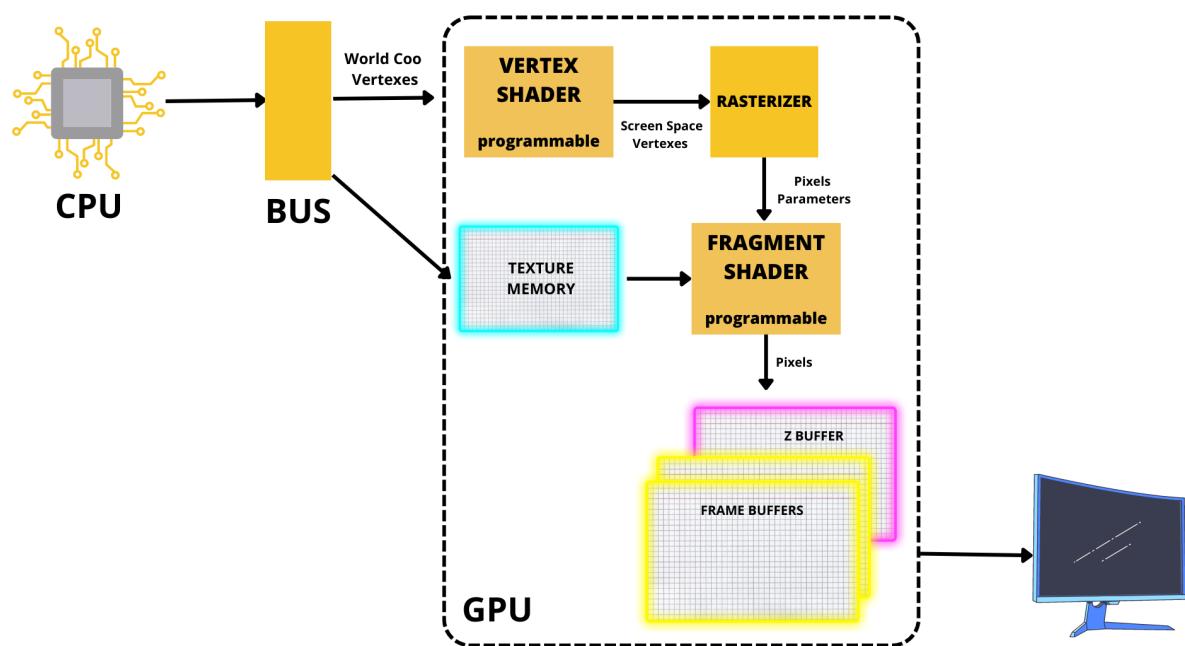


Figura 1.1: Architettura della pipeline di rendering.

pixel, calcolando la posizione esatta di ogni pixel sullo schermo e i relativi attributi come il colore, la profondità e le coordinate di texture.

Prima di trasformali in pixel, però opera due operazioni: il culling e il clipping.

Il culling è una tecnica che controlla l'orientamento dei triangoli, controllando se sono rivolti davanti o indietro (front o back face) e va a scartare tutti quelli che hanno un orientamento che li rende invisibili alla vista.

Il clipping, invece, controlla se i triangoli sono fuori dal punto di vista e, nel caso lo fossero, li va a scartare. Attraverso queste due tecniche vengono eliminati tutti i vertici che non devono essere convertiti in pixel.

Successivamente i pixel colorati vengono passati al fragment shader che, ricevuti i pixel dal rasterizzatore crea dei frammenti che contengono la posizione nello spazio dello schermo e tutte le informazioni precedentemente ottenute durante la fase di vertexing. I frammenti poi vengono trasformati in un valore di profondità e in un insieme di valori di colore che andranno scritti sui framebuffers e sul depth buffer.

I framebuffers e il depth buffer sono delle celle di memoria di transito dove vengono scritte le informazioni del fragment shader per poi essere rappresentate sullo schermo. Il framebuffer contiene le informazioni sul colore del pixel e la sua dimensione dipende dalla risoluzione dell'output. Mentre il depth buffer contiene la coordinata z e quindi la profondità di ogni pixel dell'immagine rispetto alla telecamera virtuale, confronta la profondità di ogni oggetto nella scena per determinare quali oggetti sono in primo piano e quali sono in secondo piano. Ciò permette di renderizzare l'output in modo corretto, facendo sì che gli oggetti non vengano disegnati in modo errato e sovrapposti in modo inappropriate.

Uno dei punti critici della pipeline di rendering è il trasferimento dei dati dalla CPU alla scheda grafica, quindi nel BUS: la velocità del bus può essere un fattore che limita la quantità di dati che possono essere scambiati tra i due componenti. Si verifica il fenomeno detto bottleneck o collo di bottiglia, che rallenta il flusso di lavoro del rendering, impedendo al sistema di raggiungere la massima efficienza.

Per avere una visione d'insieme migliore, si prende in analisi un'applicazione web che visualizza un oggetto, la cui vista può essere modificata interattivamente dall'utente.

HTML viene utilizzato principalmente per definire la struttura della pagina web e per

includere il codice JavaScript che implementa la logica dell'applicazione. In particolare, l'elemento canvas di HTML viene utilizzato per creare un'area di disegno sulla pagina web in cui è possibile renderizzare grafica 3D utilizzando WebGL.

```

1 <!doctype html>
2 <html>
3   <head>
4     <style>
5       body {
6         background: #fefefe ;
7       }
8       #my_Canvas {
9         padding: 10px 10px;
10        background: #ffffff ;
11        display: flex;
12        justify-content: center;
13      }
14    </style>
15  </head>
16  <body>
17    <canvas width="1250" height="570" id="my_Canvas"></canvas>
18  </body>
19  <script src="main.js"></script>
20 </html>
```

Listing 1.2: File HTML della scena presa come esempio

Più nel dettaglio viene definito lo stile del componente HTML canvas e creata un'area di disegno, dove il contesto WebGL andrà a renderizzare la scena.

Il codice JavaScript presente in *main.js*, invece, viene utilizzato per implementare la logica dell'applicazione e le interazioni con l'utente.

La libreria WebGL viene utilizzata per disegnare la scena 3D sul canvas HTML utilizzando l'accelerazione hardware della GPU. Il codice JavaScript utilizza le funzioni della libreria WebGL per inizializzare il contesto WebGL, caricare i dati della geometria dell'oggetto, vertici e indici, definire gli shader program necessari per disegnare la scena, inserendoli nel contesto di WebGL.

```

1 var canvas = document.getElementById('my_Canvas');
2 gl = canvas.getContext('experimental-webgl');
3 // Vertici della piramide
4 var vertices = [
5   // base
6   -1, 0, -1,
7   -1, 0, 1,
```

```

8   1,  0,  1,
9   1,  0, -1,
10 // vertice superiore
11 0,  1.5,  0
12 ];
13 // Definizione degli indici dei triangoli
14 var indices = [
15   0,  1,  2,
16   0,  2,  3,
17   4,  0,  3,
18   4,  1,  0,
19   4,  2,  1,
20   4,  3,  2
21 ];
22
23 ...
24
25 // Creazione gli shader program
26 const vertexShaderCode =
27   attribute vec3 position;
28   uniform mat4 Pmatrix;
29   uniform mat4 Vmatrix;
30   uniform mat4 Mmatrix;
31   attribute vec3 color;
32   varying vec3 vColor;
33   void main(void) {
34     gl_Position = Pmatrix * Vmatrix * Mmatrix * vec4(position, 1.5);
35     vColor = color;
36   } ;
37
38 const fragmentShaderCode =
39   precision mediump float;
40   varying vec3 vColor;
41   void main(void) {
42     gl_FragColor = vec4(vColor, 1.);
43   } ;
44
45 ...
46
47 const vertexShader = gl.createShader(gl.VERTEX_SHADER);
48 gl.shaderSource(vertexShader, vertexShaderCode);
49 gl.compileShader(vertexShader);
50
51 const fragmentShader = gl.createShader(gl.FRAGMENT_SHADER);
52 gl.shaderSource(fragmentShader, fragmentShaderCode);
53 gl.compileShader(fragmentShader);
54
55 const shaderProgram = gl.createProgram();

```

```

56 g1.attachShader(shaderProgram, vertexShader);
57 g1.attachShader(shaderProgram, fragmentShader);
58 g1.linkProgram(shaderProgram);

```

Listing 1.3: File JS della scena presa come esempio

Gli shader program vengono utilizzati dalla libreria WebGL per disegnare la scena. In particolare, il vertex shader viene utilizzato per trasformare le coordinate dell'oggetto dallo spazio degli oggetti allo spazio della camera, mentre il fragment shader viene utilizzato per calcolare il colore di ogni pixel della scena.

Nella scena 3D, l'oggetto viene disegnato da diversi punti di vista scelti dall'utente. Quando l'utente cambia il punto di vista, il codice JavaScript calcola la nuova posizione della camera e quindi anche la nuova matrice di vista, quindi passa queste informazioni alla libreria WebGL che utilizza queste informazioni per ricalcolare la scena dal nuovo punto di vista, per poi disegnarla sul canvas HTML.

Per disegnare sul canvas viene usata la funzione `gl.drawElements()`.

```

1 gl.drawElements(gl.TRIANGLES, indices.length, gl.UNSIGNED_SHORT, 0);

```

Listing 1.4: Funzione per disegnare gli oggetti sul canvas nella scena d'esempio.

Questa funzione utilizza gli shader program definiti in precedenza e i dati della geometria dell'oggetto, vertici e indici, per disegnare la scena sul canvas. Questo processo viene ripetuto ogni volta che l'utente cambia il punto di vista della scena, aggiornando i valori delle matrici, per garantire che la scena venga disegnata correttamente dal nuovo punto di vista.

1.3 Three.js

Three.js è una libreria JavaScript open-source che consente di creare e visualizzare grafica 3D all'interno di un browser web. Sfrutta la tecnologia WebGL per offrire un'ampia gamma di funzionalità, come la creazione di modelli 3D, l'illuminazione, l'applicazione di materiali, le animazioni e la fisica degli oggetti.

La libreria offre una serie di primitive 3D predefinite, come sfere, cilindri, cubi e mesh, che possono essere utilizzate per creare oggetti 3D. Inoltre, è possibile importare modelli 3D creati in altri strumenti, come Blender, in formato JSON, obj, glb, gltf e fbx.

Three.js supporta diverse tipologie di illuminazione per la scena, tra cui luce ambientale, puntuale, diffusa e spot. Inoltre, supporta la creazione di materiali personalizzati per simulare texture, specularità, riflessioni e altri effetti.

La libreria include un sistema di animazione interno che consente di creare animazioni di oggetti 3D e supporta la libreria di fisica Cannon.js e Ammo.js per creare effetti di fisica realistici.

L'interattività è un'altra delle caratteristiche di Three.js, supporta la navigazione all'interno della scena 3D e la selezione di oggetti. Una delle funzionalità che permette questa interazione è l'orbit controls, che, combinata al raytracer, può creare un'esperienza interattiva completa dell'utente con la scena. Ci sono anche molte librerie di terze parti disponibili che estendono le funzionalità interattive di Three.js.

Three.js è ottimizzato per funzionare su dispositivi mobili e tablet, il che significa che le tue creazioni 3D possono essere visualizzate su una vasta gamma di dispositivi. Non limita quindi i supporti di WebGL.

La libreria include inoltre, la possibilità di produrre effetti speciali, come nebbie, nuvole, particelle e ombre e utilizzare shader personalizzati.

Three.js ha una grande comunità attiva di utenti e sviluppatori che condividono risorse all'interno di un forum: sono presenti numerosi tutorial e consigli per aiutare ad imparare ed utilizzare la libreria. Inoltre, online, sono reperibili molte risorse da cui prendere spunto ed imparare.

Three.js è una scelta popolare per gli sviluppatori web poiché consente di creare contenuti 3D senza la necessità di conoscere i dettagli tecnici di WebGL. Ciò significa che gli sviluppatori web possono concentrarsi sulla creazione di contenuti 3D interattivi senza doversi preoccupare della gestione della scheda grafica del browser.

Creare un semplice scena in Three.js non è complicato. Come prima cosa è necessario iniziare creando una nuova istanza di *THREE.Scene*, che rappresenta la scena principale. Successivamente, è necessario creare una nuova istanza di *THREE.Camera*, che rappresenta la visuale della scena. Infine, è necessario creare un renderer utilizzando *THREE.WebGLRenderer* per disegnare la scena su un elemento HTML. Per aggiungere oggetti alla scena, è possibile utilizzare il metodo *scene.add(oggetto3D)*. Per creare un oggetto

3D è possibile utilizzare funzioni come *THREE.Mesh*, *THREE.BoxGeometry* rispettivamente per una Mesh oppure per un cubo e *THREE.DirectionalLight* per la creazione di una luce. Per animare la scena, si utilizza il metodo *renderer.render(scena, camera)* dentro un ciclo *requestAnimationFrame*.

```

1 import * as THREE from 'three.js';
2
3 const scene = new THREE.Scene();
4 const camera = new THREE.PerspectiveCamera( 75, window.innerWidth / window.innerHeight ,
5   0.1, 1000 );
6
7 const renderer = new THREE.WebGLRenderer();
8 renderer.setSize( window.innerWidth, window.innerHeight );
9 document.body.appendChild( renderer.domElement );
10
11 const geometry = new THREE.BoxGeometry( 1, 1, 1 );
12 const material = new THREE.MeshBasicMaterial( { color: 0x00ff00 } );
13 const cube = new THREE.Mesh( geometry, material );
14 scene.add( cube );
15
16 const directionalLight = new THREE.DirectionalLight( 0xffffff, 0.5 );
17 scene.add( directionalLight );
18
19 camera.position.z = 5;
20
21 function animate() {
22   requestAnimationFrame( animate );
23
24   cube.rotation.x += 0.01;
25   cube.rotation.y += 0.01;
26
27   renderer.render( scene, camera );
28 }
29
30 animate();

```

Listing 1.5: Creazione di una semplice scena 3D con Three.js

1.4 I principali concetti di modellazione 3D

Il concetto alla base della modellazione 3D risiede nella mesh.

Una mesh 3D è una struttura composta da vertici, lati e facce che descrivono la geometria

e la topologia di un oggetto tridimensionale. Le mesh sono utilizzate per creare modelli 3D di oggetti, personaggi, edifici e altri elementi in una scena 3D.

In generale, le facce delle mesh 3D più utilizzate sono triangoli, infatti oltre ad essere adeguati per descrivere qualsiasi forma, a differenza di quadrilateri o poligoni in genere, sono sicuramente piani e per questo semplici da gestire e renderizzare.[2]

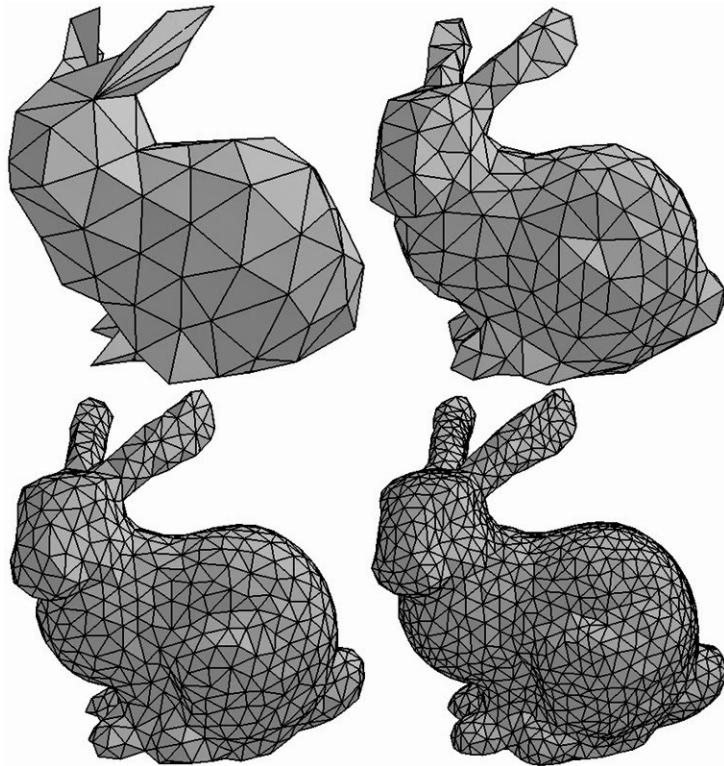


Figura 1.2: Visualizzazione di una mesh 3D composta da triangoli con diversi livelli di dettaglio.

La creazione di una mesh 3D consiste nel creare una serie di triangoli che descrivono la forma dell’oggetto. I software di modellazione più utilizzati sono Blender, 3Ds Studio Max e Maya, ma per la creazione dei nostri modelli si è optato per Blender, dal momento in cui risulta essere open-source e lo avevamo già incontrato durante il percorso di studi.

A tal proposito, risulta essere interessante descrivere la creazione di un materiale in Blender, questo può aiutare a comprendere la modifica e la creazione di alcuni dei materiali che saranno utilizzati all’interno della scena virtuale 3D.

In Blender, i materiali sono utilizzati per dare alle mesh 3D un aspetto realistico e per controllare come la luce si riflette e si diffonde su di essi. Un materiale in Blender può essere composto da diversi elementi, come texture, riflessioni, trasparenze e proprietà

fisiche della luce.

Per creare un materiale in Blender, si può utilizzare l'editor dei materiali, che si trova nella scheda "Material" della barra laterale a destra dello schermo. Nell'editor dei materiali si possono creare nuovi materiali, modificare quelli esistenti e assegnare i materiali alle mesh. In Blender ci sono diverse opzioni per creare materiali, si possono utilizzare textures e immagini per creare materiali realistici, con la possibilità di poterli combinare.

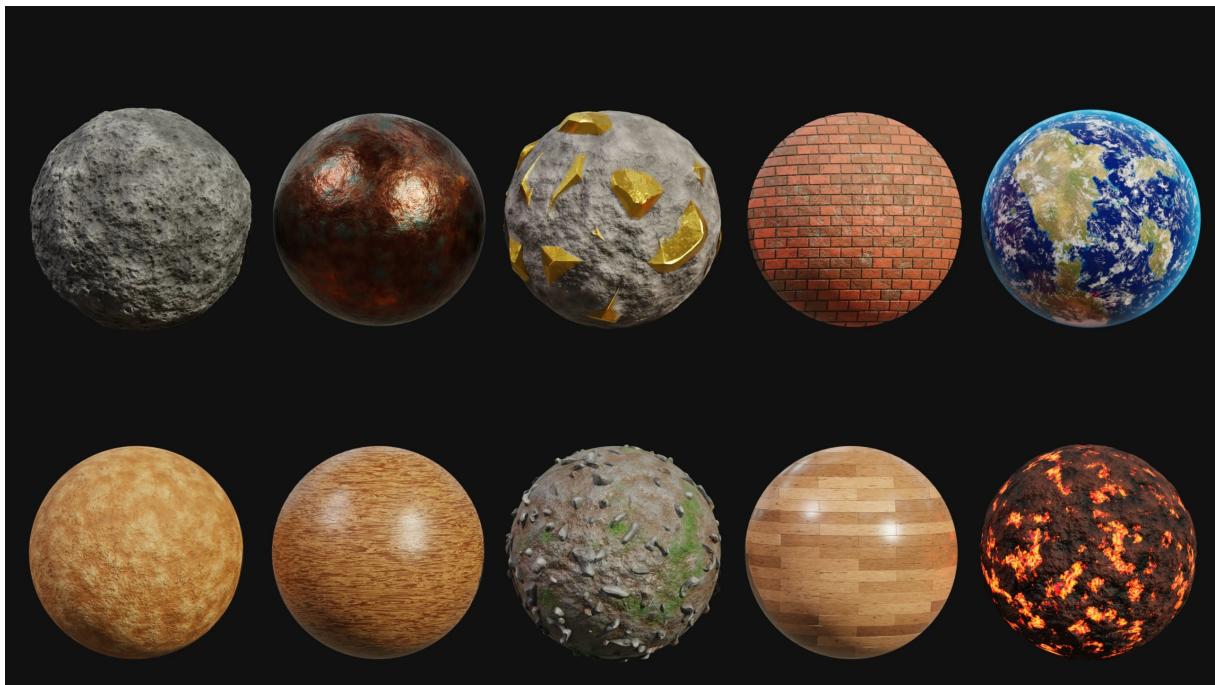


Figura 1.3: Materiali che è possibile creare in Blender, combinando texture e le proprietà del materiale.

1.5 Attività di tirocinio ed obiettivi

L'OP Group s.r.l. è un'azienda del bolognese che da oltre 50 anni si occupa principalmente di produrre espositori per cosmetici. Il processo di vendita comincia con la produzione di un modello 3D del prodotto per poi consegnare al cliente dei primi render statici.

Per migliorarlo, l'azienda ha sviluppato internamente un software che, sfruttando le librerie di three.js, consentisse di visualizzare in una stanza virtuale il loro prodotto.

La beta del software ha riscosso successo tra i clienti, mostrando ottime potenzialità, ma con la presenza ancora di grossi limiti, come: la velocità di caricamento, la resa grafica

ancora lontana da quella presente nei render tradizionali, un’interfaccia di interazione non particolarmente intuitiva e una responsività non ottimale.

Il limite più grande del software, però, risulta essere nella complessità della creazione delle scene virtuali. Infatti, sono necessarie competenze specifiche per poter sviluppare e produrre una scena, dalla creazione del modello, fino alla pubblicazione online.

L’attività di tirocinio si è posta come obiettivo quello di migliorare ed ottimizzare il software nelle criticità precedentemente citate ed inoltre, esplorare alternative che potessero fornire un upgrade del software.

L’obiettivo del tirocinio è stato quello cercare di migliorare il software sviluppato internamente ed eventualmente percorrere strade alternative che potessero permettere di creare delle scene di alta qualità grafica e con ottime performance.

1.6 Analisi del software

La pipeline di sviluppo di una scena virtuale parte dalla creazione del modello 3D in Blender, per poi importarlo all’interno del software e pubblicarlo online.

1.6.1 Creazione del modello 3D

Il primo processo di creazione del modello 3D viene fatta su 3DS Max, un software di modellazione 3D, il quale non ha la feature per poter esportare il modello in glb o gltf, ovvero i formati necessari al software. A tal proposito va utilizzato in aggiunta, Blender, software gratuito di modellazione 3D, per ultimare il processo di creazione e convertire al formato corretto il modello.

Data la complessità degli espositori e il numero elevato di prodotti al loro interno, il modello risulta essere molto dettagliato e pesante, portando il numero di mesh e triangoli a quantità difficilmente elaborabili da una macchina dalle performance nella media.

Dove sono presenti oggetti con la stessa mesh è utilizzato un sistema di template e placeholder.

Il template risulta essere una mesh che, se presenta altre istanze all’interno del modello, viene utilizzata come mesh da copiare: per ognuna delle altre copie nel modello, vengono

posizionati degli empty, chiamati placeholder, che sono dei punti senza geometria che indicano una coordinata nello spazio.

Il software, durante il caricamento della scena 3D, attraverso un accoppiamento di nomi, va a copiare la mesh del template nei relativi placeholder. In questo modo viene ridotto notevolmente il peso del modello da caricare, aumentando però il tempo di caricamento della scena.

Riguardo i materiali delle mesh, si è cercato di averne il meno possibile, e dove necessario è stato utilizzato l'UV Mapping, tecnica che mappa lo spazio 3D sullo spazio 2D, interpolando la posizione dei punti verso un punto/asse/piano attraverso una superficie.[4]

1.6.2 Funzionamento del software

Caricato l'oggetto 3D su three.js attraverso la funzione `gltfLoader()`, viene esaminata attraverso la funzione `traverse()`. Una volta divise le mesh dagli empty, i placeholder vengono sostituiti dai template e costruita la scena virtuale.

Vengono inoltre importate luci e camere ed applicati filtri per migliorare l'illuminazione e i colori degli oggetti.

L'utente, attraverso l'utilizzo del mouse, può muovere la camera al suo interno eseguendo zoom, rotazioni e pan.

Tramite la funzionalità di three.js del raycaster, si può interagire con i prodotti presenti negli espositori: passandoci sopra con il mouse, questi vengono colorati di giallo e con un doppio click del mouse vengono visualizzati singolarmente per guardarli più nel dettaglio.

Il raycaster è una classe di three.js utilizzata per il mouse picking: sfrutta la posizione del mouse per capire su quale oggetto 3D si è posizionato.[19]

Il modello 3D creato, risulta essere, nonostante tutti gli accorgimenti presi, molto pesante. A tal proposito si è pensato all'utilizzo dei LOD (Level Of Detail) che permette di mostrare mesh con più o meno dettagli in base alla distanza dalla telecamera.

Ogni livello è associato ad un oggetto, e il rendering può essere commutato tra di loro alle distanze specificate. In genere vengono create, per esempio, tre mesh, una per il lontano (dettaglio basso), una per la gamma media (dettaglio medio) e una per i primi piani (dettaglio elevato).[20]

Infine, sono presenti algoritmi di post-processing che mirano a migliorare la resa grafica della scena aggiungendo ombre ed effetti che portino alla verosimilità i materiali. Vengono aggiunti anche algoritmi per ridurre l'aliasing presente sui contorni delle mesh.

1.6.3 Ricerca di metodi e strumenti per migliorare il servizio aziendale

Una delle problematiche principali riguardava la resa dei materiali: i vetri non avevano la trasparenza adatta, gli specchi non producevano riflessione, i prodotti metallici risultano piatti e non riflettenti e infine, alcuni dei prodotti di cartone non avevano l'opacità adeguata.

Attraverso la modifica dei materiali in Blender è stato possibile, giocando con i parametri principali, ottenere miglioramenti sostanziali rispetto alla base di partenza.

Riguardo la trasparenza dei vetri, è stata ridotta al minimo la Roughness, aumentata al massimo la Transmission e posto il valore dello Specular a 0.5.

Per ottenere il riflesso dei vetri è stato necessario aumentare al massimo il Metallic e lo Specular e ridurre al minimo la Roughness. Il risultato ottenuto non risulta essere ottimale, ma un buon trade-off, dal momento in cui viene specchiato solamente l'environnement. Il risultato ideale sarebbe stato quello di veder specchiati anche le mesh della scena 3D, ma questo è ottenibile solamente attraverso un algoritmo di post processing chiamato Screen Space Reflections.

I materiali metallici invece, sono stati migliorati modificando i parametri di Metalness e Specular e riducendo in base al livello di riflessione che si voleva ottenere, il parametro di Roughness. Infine, per ottenere la miglior resa sui materiali di cartone, è stato ridotto al minimo il parametro di Transmission e aumentato quello di Roughness.

Già con queste modifiche la resa dei materiali è risultata essere migliorata e molto simile a quello che è il target di resa dell'azienda.

La scena virtuale, inizialmente, presentava 6 mobili, popolati con centinaia di prodotti, un insieme di circa 25 camere, luci, e una stanza d'ambiente.

Nonostante gli accorgimenti per ottimizzare le prestazioni della scena, questa mole di oggetti da renderizzare porta inevitabilmente ad appesantire il rendering e ad aumentare

i tempi di caricamento e di risposta.

Dopo un confronto è emerso che visualizzare tutto il corner di un negozio in una scena non è fondamentale. A tal proposito, un banale accorgimento è quello di semplificare il più possibile la scena.

Infatti, al cliente potrebbe essere proposto solamente un singolo mobile con i prodotti al suo interno, senza l'aggiunta di camere per ripiano e di particolari luci per illuminare tutta la scena. La riduzione di mesh e quindi di triangoli, di camere, luci, materiali e texture garantisce un miglioramento di tutte le performance.

Un altro degli scogli principali dell'azienda è quello della pipeline di sviluppo delle scene 3D, a partire dalla creazione del modello fino al caricamento online.

La creazione del modello, per il reparto grafico dell'azienda risulta essere tedioso e dispendioso in termini di tempo, infatti, dopo aver creato il modello 3D per i render statici, è necessario convertirlo in quello idoneo alla scena virtuale, eliminando tutte le mesh doppie e sostituendole con degli empty. Inoltre, caricare il modello all'interno del codice non risulta essere immediato.

Sarebbe quindi necessario trovare un pipeline di sviluppo più semplice e ottimizzata. Inizialmente si potrebbe sostituire il modello di template-placeholder con un semplice oggetto 3D, al cui interno vi sono tutte le mesh. Questo è ottenibile solo se la scena virtuale che si vuole visualizzare non eccede in quantità di mesh e triangoli, come quella iniziale.

Ciò andrebbe a semplificare il lavoro del reparto grafico, dovendosi vincolare solo ad attenzioni sulla nomenclatura delle mesh e dei materiali.

Su richiesta è stato proposto di aggiungere un'infografica dei prodotti nella modalità focus. È stata creata attraverso l'utilizzo di CSS per modificare lo stile dei componenti di HTML. Ogni qual volta viene cliccato due volte su un prodotto, in base al nome della mesh, vengono modificati dinamicamente i dati, a partire dal titolo, fino alla descrizione del prodotto. Dove possibile è stata aggiunta anche la possibilità di poter cambiare il colore del materiale come ad esempio il colore della pasta di una tipologia di rossetto.

In ultima analisi è stata intrapresa una ricerca per valutare alternative esterne al software sviluppato. Sono state trovate alcune soluzioni che permettono di visualizzare modelli 3D, ma senza particolari feature aggiuntive. Tra i principali figura gltf-viewer: framework

open-source che permette di caricare modelli in formato glb e gltf. Consente di modificare molti parametri tra cui: environment, luci, camere, wireframe e visualizzazione dei colori.

Non fornisce però molte delle feature che porterebbero un sostanziale miglioramento alla base di partenza: gli algoritmi di post-processing.

Dopo aver svolto all'interno dell'azienda attività volte a comprendere il workflow di produzione, è stato proposto come progetto di tesi lo studio e l'analisi di un software mirato a migliorare le prestazioni e la qualità di rendering dei loro progetti.

Il software in questione è WebGi, ed è rilasciato in versione beta come un ambiente di sviluppo (SDK).

Capitolo 2

WebGi SDK

In questo capitolo viene descritto il funzionamento di WebGi, entrando nel dettaglio sui plugin principali che rappresentano il core del sistema.

WebGI è un framework, scritto in TypeScript, che consente di visualizzare modelli 3D basato su rendering foto realistici e facilità di utilizzo dello sviluppatore. Attualmente è ancora in open beta e negli ultimi mesi ha ricevuto aggiornamenti costanti fino ad arrivare alla versione 0.6.6.

E' studiato per essere utilizzato con facilità sui siti web, ha molte funzionalità a disposizione e, essendo basato su three.js, consente di applicare modifiche facilmente per renderlo idoneo a tutte le necessità. Le tecnologie alla base di WebGi permettono di scaricare un file HTML da incorporare direttamente su un sito web, oppure all'interno di una applicazione.

Per garantire il foto realismo utilizza algoritmi di post-processing come SSR, SSAO, anti-aliasing, Bloom e refraction. Inoltre, ha librerie di materiali realizzati per visualizzare al meglio alcuni dei materiali più complessi da ottenere come pietre preziose, tessuto, glitter, vernice per auto, pelle e metalli.

Sul sito di WebGi è possibile accedere all'editor online, dove, senza scaricare niente è possibile lavorare direttamente sulle scene create, con animazioni e materiali personalizzabili, andando a modificare ogni tipo di configurazione. Grazie alla possibilità di esportare le scene, anche ottimizzandole attraverso algoritmi di compressione, è possibile creare la propria scena virtuale online per poterla esportare ed utilizzare all'interno del software e di conseguenza nei siti web e nelle applicazioni.[25]

2.1 Strumenti per l’uso di WebGi

2.1.1 Node.js

Node.js è una piattaforma JavaScript open-source che consente di eseguire il codice JavaScript sul lato server. È stato progettato per creare applicazioni ad alte prestazioni e scalabili, utilizzando JavaScript su entrambi i lati, front-end e back-end.

Node.js utilizza un’architettura a eventi asincroni, che consente di gestire molti utenti contemporaneamente senza rallentamenti. Questo rende Node.js particolarmente adatto per le applicazioni ad alte prestazioni, come i giochi in tempo reale e di conseguenza anche scene virtuali 3D.

Node.js include un motore JavaScript V8 sviluppato da Google e una libreria di moduli JavaScript chiamata "npm" (Node Package Manager), che consente di installare e gestire facilmente le dipendenze delle applicazioni.

Per poter utilizzare WebGI, attualmente è necessaria la versione 14 o una versione superiore, facendo attenzione ad aggiungere tutte le dipendenze necessarie nella fase di installazione.

2.1.2 Editor di codice

Per poter integrare WebGi è necessario un editor di codice, per poter modificare script scritti in javascript e typescript. I due editor consigliati sono: Visual Studio Code e JetBrains WebStorm.

2.1.3 GitHub

GitHub è una piattaforma di sviluppo web che offre hosting per progetti software basati su Git. Git è un sistema di controllo versione che consente ai team di sviluppo di lavorare sullo stesso codice contemporaneamente, di tenere traccia delle modifiche e di gestire le versioni del proprio progetto.

Con GitHub, gli utenti possono creare repository (spazi di archiviazione online) per i propri progetti, invitare altri utenti a collaborare e condividere il codice con la comunità. GitHub offre anche funzionalità avanzate per la gestione del codice, come la possibilità

di creare branch del codice per sviluppare funzionalità in parallelo, creare e gestire pull request per integrare le modifiche degli utenti e utilizzare gli strumenti di gestione delle attività per organizzare il lavoro.

Dalla repository GitHub di WebGi è possibile scaricare il template che fornisce il codice iniziale per poter iniziare ad usarlo.

2.1.4 Parcel

Il templete utilizza Parcel bundler web open-source che rende semplice l'utilizzo di tecnologie web moderne come JavaScript, CSS e HTML.

Permette agli sviluppatori di creare progetti web senza la necessità di configurare manualmente un ambiente di sviluppo.

Con Parcel, gli sviluppatori possono creare un progetto web semplicemente creando un file HTML e iniziando a scrivere codice. Parcel si occupa automaticamente della gestione dei moduli, della conversione dei file e dell'ottimizzazione delle prestazioni.

Una volta integrati questi strumenti, attraverso il comando *npm install* si installano tutte le dipendenze necessarie per utilizzare WebGi e con il comando *npm run start* si fa la build del progetto e viene lanciato il development server.

2.2 La struttura di WebGi

WebGi è fornito di una pipeline di rendering e molti plugin, che possono essere aggiunti attraverso una sola riga di codice, questo fornisce una vasta varietà di funzionalità. Infatti, è possibile creare un'applicazione personalizzata ottimizzando il bundle scegliendo solo i plugin necessari.

Risulta interessante fare un breve confronto fra la struttura di three.js e quella di WebGi. Come detto in precedenza, three.js è una libreria scritta in JavaScript che permette la creazione di scene virtuali in realtime e ci si può avvalere inoltre, di altre librerie di supporto per semplificarne la creazione. WebGi fornisce plugin scritti in TypeScript; si può pensare ad un plugin di WebGi come ad una libreria di supporto in three.js.

TypeScript è un superset di JavaScript che fornisce funzionalità avanzate come la tipizzazione statica, le classi e l'ereditarietà. Un plugin o una libreria sviluppati in TypeScript possono essere compilati in JavaScript per essere utilizzati in ambienti che non supportano nativamente TypeScript. Questo significa che un plugin o una libreria scritta in TypeScript può essere utilizzato come se fosse una libreria JavaScript standard, ma con il vantaggio della tipizzazione.

A tal proposito si può pensare a WebGi come ad una libreria come three.js, ma di più alto livello, poiché presenta un insieme più ampio di costrutti ed astrazioni che rendono più facile per un programmatore scrivere codice efficiente e mantenibile. Segue ora un'analisi più dettagliata dei plugin sviluppati da WebGi.

2.2.1 3D Viewer Core

Il core di WebGi è costituito dal viewer 3D, che è sviluppato sulla base di Three.js, in una versione leggermente modificata dallo sviluppatore. Essa risulta essere accessibile e consultabile.

Il viewer è ottimizzato per tutti i dispositivi, è infatti possibile utilizzarlo da smartphone, tablet e anche dispositivi obsoleti.

Questo è permesso da due features: Adaptive Quality Control e Progressive Rendering Pipeline.

L'Adaptive quality control è una tecnica di ottimizzazione del rendering che consente di adattare la qualità del rendering in base alle esigenze dell'applicazione. Vengono monitorate continuamente le prestazioni dell'applicazione e di conseguenza modificate dinamicamente la quantità di dettagli e la complessità del rendering in modo da mantenere un equilibrio tra qualità e prestazioni.

Questa tecnica è particolarmente utile per le applicazioni che eseguono il rendering in tempo reale, in quanto permette di garantire un'esperienza fluida e reattiva per l'utente, indipendentemente dalle prestazioni del sistema. Ad esempio, se l'applicazione sta eseguendo il rendering in un sistema con un basso potenziale di elaborazione, l'Adaptive Quality Control può ridurre la quantità di dettagli nel rendering per mantenere le prestazioni accettabili.

Inoltre, il controllo della qualità adattiva è anche utile per le applicazioni che richiedono

una grande quantità di dettagli solo in alcune aree della scena, ad esempio, in prossimità dell’utente o di un oggetto selezionato. In questi casi può aumentare la quantità di dettagli solo nelle aree interessate, mantenendo al contempo prestazioni accettabili per il resto della scena. In particolare vanno ad influire la risoluzione dello schermo, la capacità di calcolo del dispositivo e la banda di rete.

Più nel dettaglio, l’Adaptive Quality Control combina alcune tecniche come lo scaling della risoluzione, la compressione dei dati, la selezione del bitrate e l’adattamento dinamico della qualità.

Lo scaling della risoluzione consiste nell’adattare il contenuto alla risoluzione dello schermo del dispositivo. Ad esempio, se un dispositivo mobile ha una risoluzione dello schermo più bassa rispetto ad un dispositivo desktop, il sistema di Adaptive Quality Control può ridurre automaticamente la risoluzione del contenuto per adattarsi alla capacità del dispositivo.

Questo processo è detto downscaling: viene ridotto il numero di pixel dell’immagine attraverso il sub-scaling o sotto-campionamento, ovvero un metodo che sceglie i pixel da tenere attraverso algoritmi come il nearest neighbour, quindi il pixel più simile a quello selezionato oppure ad esempio attraverso il calcolo della media. Il processo inverso è l’upscale o sovra-campionamento che, attraverso l’interpolazione, un metodo in cui i pixel dell’immagine vengono duplicati e interpolati per creare nuovi pixel fino a raggiungere la risoluzione desiderata, anche in questo caso ci sono molti algoritmi, tra cui alcuni che utilizzano funzioni matematiche complesse o tecniche di machine learning.

Un’altra tecnica utilizzata dall’Adaptive Quality Control è la compressione dei dati che permette di ridurre le dimensioni dei file ed aumenta la velocità di trasmissione dei dati. La compressione viene effettuata in maniera adattiva, in base alle performance del dispositivo di destinazione, alla banda di rete e soprattutto in base alla complessità dell’immagine. Più è complessa l’immagine e meno performante il dispositivo, più grande sarà la compressione che verrà effettuata.

Infine, l’Adaptive Quality Control adotta l’adaptive bitrate, per bitrate si intende la quantità di dati che vengono trasmessi al secondo e può essere regolato in base alla larghezza di banda disponibile. Ad esempio, se la larghezza di banda è bassa, il sistema di Adaptive Quality Control può diminuire il bitrate per garantire un render più fluido dell’immagine.

La Progressive Rendering Pipeline è una tecnica di rendering che prevede un render incrementale, progressivo dell’immagine, aggiungendo dettagli passo passo.

La pipeline di progressive rendering si basa generalmente sull’utilizzo di algoritmi come il path tracing, detto anche random Path Tracing di Monte Carlo o il Progressive photon mapping, tecniche per cui i raggi della luce vengono tracciati dalla sorgente luminosa per calcolare l’illuminazione di ogni punto dell’immagine.[21]

Inizialmente, il rendering viene eseguito con una risoluzione bassa e con una quantità limitata di campionamenti per ogni pixel. Man mano che il rendering procede, la risoluzione e la quantità di campionamenti vengono gradualmente aumentati, migliorando la qualità dell’immagine.

Ci sono molte tecniche di progressive rendering, ma seguono tutte dei passi come linea generale. Il primo di questi è il caricamento della scena per poter fare il rendering, quindi le geometrie con i materiali e le texture e le luci, successivamente viene generata un’immagine a bassa risoluzione campionando un numero limitato di pixel e producendo un primo render, che rappresenta un’approssimazione della scena finale. In seguito vengono incrementati progressivamente i campionamenti per ogni pixel, migliorando il dettaglio e, ogni qual volta vengono fatti i campionamenti, l’immagine renderizzata viene aggiornata. Qui si possono intraprendere diverse strade, la prima di queste è interrompere il rendering progressivo, quando si raggiunge un numero massimo di campionamenti, oppure, tramite l’Adaptive Quality Control, controllare che il carico computazionale del rendering non crei rallentamenti delle performance.

Parlando della resa delle immagini, il viewer sviluppa l’HDR RGBM pipeline. Tale processo di rendering consente di ottenere immagini HDR (High Dynamic Range) all’interno di una applicazione 3D. Il formato HDR RGBM (Red Green Blue Multi-Channel) è un formato di codifica per immagini di alta qualità, come quelle HDR.[5] Questo formato utilizza più canali di colore per rappresentare una gamma più estesa di luminosità rispetto ai formati ad 8-bit.

L’HDR RGBM permette di rappresentare valori di luminosità più alti e più bassi rispetto ad altri formati, permettendo di catturare dettagli sia nelle ombre che nei punti luminosi della scena.

Per realizzare questa pipeline è necessario convertire i dati dell’immagine HDR in formato

RBGM; la conversione risulta essere in realtà una compressione, dal momento in cui il formato RGBM è un formato meno pesante in termini di memoria. Infatti, viene utilizzato un canale aggiuntivo oltre i classici RGB, che è il canale "M": è un canale a bassa precisione dove vengono salvate le informazioni relative alla luminosità. Questo tipo di approccio permette di ottimizzare l'utilizzo della memoria e di ridurre il costo computazionale, cercando un compromesso tra la resa grafica di un'immagine HDR e delle buone prestazioni. Risulta infatti essere molto utilizzata nelle applicazioni web e per applicazioni su dispositivi mobile.

Un processo simile viene effettuato dall'HDR Half-Float/Float + sRGB pipeline. Analogamente all'HDR RGBM pipeline, le immagini HDR vengono codificate, ma in questo caso, nel formato Half-Float/Float. Quindi utilizzando rispettivamente 16 e 32 bit per rappresentare i valori di luminosità. L'immagine viene elaborata in floating/half floating point e poi convertita in sRGB per la visualizzazione finale. Questo approccio offre una migliore qualità dell'immagine rispetto alla HDR RGBM, ma richiede anche un maggiore costo computazionale e di memoria.[10]

In entrambi i casi, l'obiettivo è di fornire una rappresentazione HDR più realistica delle immagini, ma la scelta di quale pipeline utilizzare dipende dalle esigenze specifiche dell'applicazione e dalle limitazioni hardware disponibili. Il vantaggio nell'utilizzo di queste due pipeline permette di ottimizzare la visualizzazione delle immagini HDR, scegliendo quale pipeline è più adatta per un dispositivo o per una particolare situazione.

A livello architetturale il viewer utilizza una Simple API con un Single Entry Point, utilizza per cui un API gateway che gli permette di accedere a tutte le funzioni ed operazioni possibili attraverso un'unica interfaccia. Ciò garantisce sicurezza e semplifica la modifica e manutenzione del codice da parte degli sviluppatori, avendo da controllare solo un entry point.[15]

Infine, per concludere le funzionalità del core, il viewer implementa il Jittered Shadow Mapping, producendo mappe d'ombra di alta qualità quando vengono utilizzate luci direzionali o luci spot.

L'implementazione di questo plugin mira ad eliminare i problemi che possono crearsi dall'utilizzo delle shadow map come il flickering o il banding.

Il jittering consiste nell'aggiungere una quantità di rumore casuale alle coordinate di

campionamento delle shadow map, prima che essa venga calcolata. Questo ha l'effetto di distribuire i campioni delle ombre in modo casuale sulla superficie dell'oggetto illuminato, prevenendo la formazione di pattern regolari che causano il banding, ovvero il fenomeno per cui una transizione graduale di colore su una superficie appare come una serie di bande di colore distinte, invece di essere fluida e uniforme.

Inoltre, l'aggiunta dei jitter può anche ridurre il flickering delle ombre. Il flickering si verifica quando i campioni delle ombre non sono perfettamente allineati tra un'immagine e l'altra, il che può essere causato da piccole variazioni nella posizione della fonte di luce o degli oggetti illuminati. L'aggiunta di rumore casuale alle coordinate di campionamento rende più difficile la visualizzazione di queste variazioni minime, riducendo il flickering delle ombre. In tal modo, l'utilizzo del Jittered Shadow Mapping permette di ottenere ombre migliori, andando a ridurre gli effetti visivi indesiderati che possono verificarsi.

L'entry point principale del viewer 3D è la classe *ViewerApp* che crea un scena, un renderer e li attacca al canvas. Per cui la *ViewerApp* può essere inizializzata semplicemente attraverso canvas.

```
1 import {ViewerApp} from "webgi";
2
3 async function initialize3D () {
4     const viewer = new ViewerApp({
5         canvas: document.getElementById('main-canvas') as HTMLCanvasElement,
6     })
7 }
```

Listing 2.1: Creazione del viewer attraverso l'elemento canvas

Se non è richiesto gestire la creazione del canvas, il visualizzatore può essere istanziato passando il container: verrà creato un canvas che riempie completamente il container. Ad esempio, per creare un visualizzatore a schermo intero, va passato il body come contenitore:

```
1 const viewer = new ViewerApp({
2     container: document.body,
3 })
```

Listing 2.2: Creazione del viewer attraverso il container

Infine, è possibile aggiungere opzioni per gestire la pipeline di rendering:

```
1 const viewer = new ViewerApp({
```

```

2     canvas,
3     useRgbm: true, // Use HDR RGBM Pipeline. false will use HDR Half float pipeline
4     useGBufferDepth: true, // Uses depth prepass for faster rendering, has z-fighting in
      some cases.
5     isAntialiased: false, // Uses multi-sample render target. (only for extreme cases)
6   })

```

Listing 2.3: Creazione del viewer attraverso il canvas con opzioni per il rendering

La classe *ViewerApp* eredita la classe *SimpleEventDispatcher<"update" / "preRender" / "postRender" / "preFrame" / "postFrame" / "dispose" / "addPlugin">*. La gerarchia fra due classi può essere definita mediante l'ereditarietà. Una classe figlia può ereditare dalla classe padre ereditando i suoi membri: proprietà e metodi. In questo modo, la classe figlia può ampliare o sovrascrivere i membri della classe padre.

Il viewer è quindi una classe che estende la funzionalità di un event dispatcher, limitando i tipi di eventi che possono essere pubblicati e ricevuti. La stringa tra parentesi tonde specifica i tipi di eventi supportati dalla classe, in questo caso "update", "preRender", "postRender", "preFrame", "postFrame", "dispose" e "addPlugin".

Questo significa che l'event dispatcher può pubblicare solo questi tipi di eventi e che gli oggetti che si registrano come listener possono ricevere solo notifiche di questi eventi specifici. Ciò garantisce che gli eventi siano ben strutturati e che gli oggetti che ricevono notifiche sappiano esattamente cosa aspettarsi.

L'uso di un event dispatcher limitato in questo modo rende il codice più leggibile e più facile da mantenere, poiché gli sviluppatori hanno una chiara comprensione dei tipi di eventi che possono essere pubblicati e ricevuti dalla classe.[24]

Infine, il *ViewerApp* viene fornito con una libreria di plugin per interagire con la scena 3D, aggiungere nuove funzionalità. Ciò viene fatto attraverso uno dei metodi precedenti: *addPlugin*.

Un plugin è un componente software che estende o amplia le funzionalità di un'applicazione. Un plugin è progettato per essere integrato in un'applicazione esistente e offre nuove funzionalità o modifiche a quelle esistenti, senza richiedere modifiche al codice sorgente dell'applicazione stessa.

I plugin possono essere utilizzati per aumentare la funzionalità di un'applicazione in una vasta gamma di modi, ma in questo caso va ad aggiungere nuove funzionalità per la

creazione di contenuti.

Alcuni dei plugin principalmente usati per la creazione di una scena in WebGi saranno descritti all'interno delle features del framework.

2.2.2 Asset Management e importazione

L'Asset Manager permette di caricare modelli 3D, in molteplici formati, all'interno della scena. Il plugin AssetManager gestisce il download, la gestione, la cache, l'analisi, il caricamento e l'aggiunta di asset alla scena.

I formati che è possibile importare sono: GLTF (.gltf e .glb files, anche con draco compression), HDR (.hdr e .hdr.png files con RGBE encoding), PNG/JPG/SVG (image formats supportati dal browser), VJSON (.vjson files per il viewer e i presets di configurazione del plugin), PMAT (.pmat physical material files), JSON (.json files e presets), TXT (.txt semplici file di testo) e HDR.PNG.

Attraverso l'utilizzo di ulteriori plugin è possibile importare modelli in formato FBX e OBJ/MTL, file .zip, environment map in formato EXR.[27]

```

1 import {
2     ViewerApp,
3     AssetManagerPlugin ,
4 } from "webgi";
5 ... // your code
6 const viewer = ...
7 ... // viewer setup and other plugins.
8 const manager = await viewer.addPlugin(AssetManagerPlugin);
9 ...
10 // after adding the manager can be accessed from the viewer like this:
11 const manager = viewer.getManager();
12
13 const options = {
14     autoScale: true, // Scales the object before adding to the scene.
15     autoScaleRadius: 2, // Scales the object bounding box to 2 World Units, if
16         // autoScale is true
17     pseudoCenter: true, // centers the object to origin on load
18     // check docs for other options (if required)
19 }
20 const assets = await manager.addFromPath("path/gltf/model3D.glb", options)
21 }
```

Listing 2.4: Esempio base di utilizzo dell'asset manager plugin in WebGi.

2.2.3 Materiali e shaders

WebGi, ottimizza i materiali fisici di three.js attraverso il Physically Based Rendering (PBR) workflow. Il PBR è un insieme di tecniche di rendering che si basano sul simulare il comportamento fisico della luce nell’ambiente reale.

Poiché il rendering basato sulla fisica mira a imitare la luce in modo fisicamente plausibile, in genere ha un aspetto più realistico rispetto agli algoritmi di illuminazione originali come Phong. Questo modello di illuminazione è più complesso poiché vengono utilizzate equazioni fisiche per il calcolo dell’interazione tra luce e materiali.

Tutte le tecniche PBR si basano sulla teoria delle microfaccette. La teoria descrive che qualsiasi superficie su scala microscopica può essere descritta da piccoli specchi perfettamente riflettenti chiamati microfaccette. A seconda della rugosità di una superficie, l’allineamento di questi piccoli specchi può variare notevolmente: più una superficie è ruvida, più ogni microfaccia sarà allineata in modo caotico lungo la superficie, al contrario, su una superficie liscia è più probabile che i raggi luminosi si riflettano più o meno nella stessa direzione.

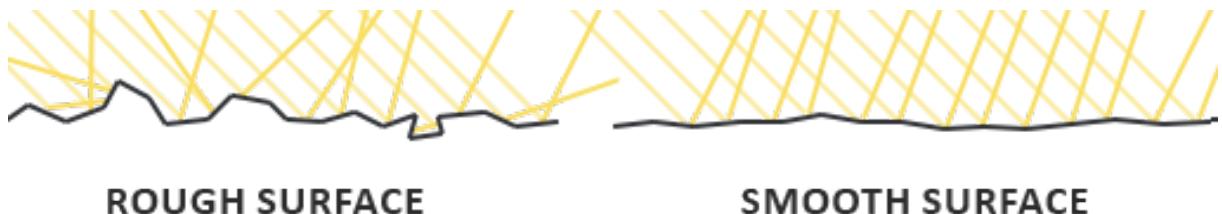


Figura 2.1: Orientamento dei raggi luminosi in base alla superficie di incidenza.

Passando ad alcune delle equazioni usate dal PBR, una delle più note è l’equazione di Fresnel, questa equazione per ottenere la riflessione R è molto complessa ed è stata fatta un’approssimazione da Schlick:

$$R = R_0 + (1 - R_0) * (1 - \cos\theta)^5$$

dove:

- $R_0 = (\frac{n_1 - n_2}{n_1 + n_2})^2$
- n_1 = IoR del mezzo di provenienza

- n_2 = IoR del mezzo di destinazione
- $\cos\theta$ = coseno dell'angolo tra il vettore normale e la direzione della luce incidente
- R = fattore di riflessione

Il fattore di riflessione è una delle proprietà che si possono personalizzare, ad esempio, all'interno di Blender quando si crea un materiale, più alta sarà, più il materiale rifletterà la luce e viceversa.

Un'altra equazione utilizzata nel PBR è l'equazione per calcolare la funzione di distribuzione riflettente bidirezionale (BRDF) di Cook-Torrance:

$$BRDF = \frac{DFG}{4(\omega_o * n)(\omega_i * n)}$$

dove:

- D = funzione di distribuzione normale che approssima la quantità di microfaccette della superficie allineate al vettore di metà strada, influenzata dalla rugosità della superficie; è la funzione principale che approssima le microfaccette
- F = l'equazione di Fresnel che descrive il rapporto di riflessione della superficie a diversi angoli di superficie
- G = la funzione geometrica che descrive la proprietà di auto-ombra dei microfaccette. Quando una superficie è relativamente ruvida, le microfaccette della superficie possono mettere in ombra altre microfaccette, riducendo la luce riflessa dalla superficie
- ω_o = è la direzione in ingresso della luce
- ω_i = è la direzione in uscita della vista
- n = è la normale della superficie

Dopo aver visto alcune delle equazioni che rappresentano parte del modello matematico del PBR, ciascuno dei parametri di superficie necessari per una pipeline PBR può essere definito o modellato da texture. L'uso delle texture ci permette di controllare per ogni

frammento il modo in cui ogni specifico punto della superficie deve reagire alla luce: se quel punto è metallico, ruvido o liscio, o come la superficie risponde alle diverse lunghezze d'onda della luce.

Le texture principali sono la normal map, la metallic map, la roughness map e l'ao (ambient occlusion) map.[12]

Dopo aver applicato il PBR, WebGi dà la possibilità di integrare la creazione e la modifica dei materiali attraverso alcuni plugin.

Il primo di questi è l'Anisotropy Plugin che aggiunge l'anisotropia del metallo al materiale PBR. L'anisotropia metallica è un effetto visivo che permette di descrivere come i metalli riflettono la luce in modo differente a seconda della direzione. I materiali più difficili da rappresentare sono il rame e l'acciaio inossidabile che, avendo una superficie ruvida, hanno un comportamento differente quando ricevono l'illuminazione.

Per realizzarla si utilizza una mappa direzionale anisotropica, ovvero una texture che controlla la forma dei riflessi speculari delle luci in relazione alla rugosità del materiale in tempo reale.[17]

Il Clearcoat Tint Plugin aggiunge Tint, Thickness e IoR al materiale PBR. Il Tint è una tecnica visiva che consente di modificare il colore di un'immagine o di un oggetto 3D applicando un colore uniforme a tutta la superficie.

Viene fatto attraverso tecniche di shading dove viene utilizzato per modificare il colore di ogni pixel dell'immagine o dell'oggetto 3D, unendo il tint al colore originale.

La thickness indica la profondità o lo spessore del materiale. Applicata ai materiali PBR serve principalmente nei materiali trasparenti e vetrati, per ottenere al meglio gli effetti visivi di trasparenza. È importante utilizzarla poiché l'interazione della luce con il materiale dipende anche dal suo spessore.

Lo IoR è l'indice di rifrazione è un valore che indica quanto un materiale influisce sulla direzione e l'intensità della luce che lo colpisce.

Gli IoR più comuni sono quelli dell'acqua, che è 1.33 e del vetro: 1.52. Poiché lo IoR del vetro è maggiore di uno, la luce che attraversa il vetro viene deviata verso l'interno e si rifrange. In generale, un materiale con un IOR più elevato avrà una rifrazione maggiore, mentre un materiale con un IOR più basso avrà una minore capacità di rifrazione.[1] Nel contesto dei materiali PBR viene utilizzato principalmente per ottenere il realismo nei

materiali come l’acqua, i vetri e i cristalli.

Riguardo il tema della rifrazione dei materiali, WebGi implementa l’High Performance Screen space Refraction, è una tecnica di rendering che simula gli effetti della rifrazione della luce attraverso superfici trasparenti o riflettenti. La rifrazione è il fenomeno per cui la direzione della luce cambia quando attraversa un mezzo con un IoR diverso.

L’High Performance Screen space Refraction si basa sull’utilizzo dello spazio schermo (screen space) per calcolare i raggi della luce riferiti attraverso la superficie di un oggetto trasparente o riflettente. Per ottenere alter performance viene sfruttata la potenza di calcolo della GPU per effettuare il rendering degli effetti di rifrazione in tempo reale, utilizzando solo le informazioni contenute nello screen space (lo spazio delle coordinate dell’immagine 2D risultante dalla proiezione 3D sulla geometria nello spazio della telecamera[28]), quindi senza la necessità di calcolare i raggi della luce in maniera completa. Questo permette di ottenere una notevole velocità di rendering, mantenendo allo stesso tempo una buona qualità visiva.

Per realizzarla si procede con il calcolo delle normali delle superfici visibili presenti nella scena. Queste informazioni vengono utilizzate per determinare l’angolo di incidenza della luce sulla superficie. Successivamente si calcolano le coordinate dello screen space per ogni pixel visibile, questo viene fatto per ottenere la posizione degli oggetti rispetto la posizione della camera. Ed infine, viene calcolata la rifrazione utilizzando le normali delle superfici e le coordinate, viene calcolata la posizione del punto di rifrazione e la direzione della luce riferita per ogni pixel. Questi valori, poi, vengono utilizzati per determinare il colore finale del pixel.

In aggiunta a questi plugin, WebGi permette di utilizzare il Noise Bump Material Plugin e Parallax Bump Mapping Plugin.

Il Noise Bump Mapping è una tecnica utilizzata per aumentare la complessità degli oggetti realizzati in 3D, senza modificare la struttura geometrica di partenza. Questo è un effetto che viene applicato per ricreare solchi e sporgenze sulle superfici senza appesantire la scena. Il Noise Bump Mapping si basa su una sovrapposizione di una seconda texture applicata sull’originale, detta Noise Bump Map, caratterizzata da una gradazione cromatica in scala di grigio: a seconda dei valori di luminanza dell’immagine, il bump creerà l’effetto desiderato facendo corrispondere alle zone della texture di colore bianco superfici con una

sporgenza maggiore ed, in quelle di colore nero, depressioni e avvallamenti.[6]

Un esempio dell'effetto ottenuto applicando una noise map ad una mesh è il seguente

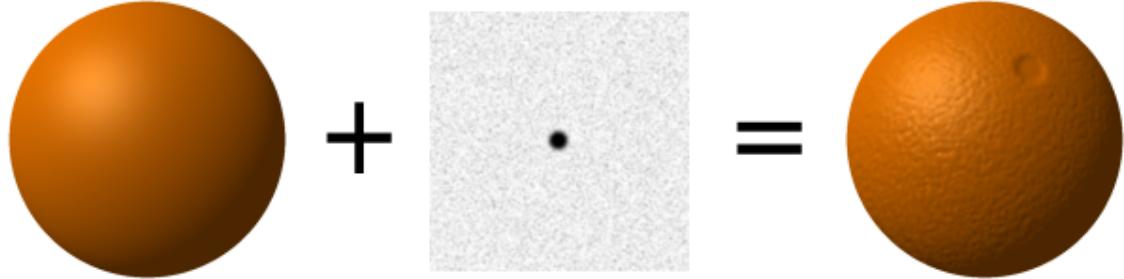


Figura 2.2: Noise Bump Mapping applicato ad una mesh sferica.

Il Parallax Bump Mapping migliora l'effetto visivo del bump mapping utilizzando il fenomeno del parallasse: l'effetto per cui un oggetto sembra spostarsi rispetto allo sfondo, se si cambia punto di osservazione. Per l'utente finale questo significa che alcune texture, ad esempio la rappresentazione di un muro in pietra, avranno una profondità più evidente e quindi maggiore realismo, il tutto con una minore influenza sulle prestazioni della simulazione.[29]

Il Parallax Bump Mapping riesce a gestire direttamente le informazioni di mappatura sul singolo pixel ed, in base ad esse ed all'angolo di visuale, modifica le coordinate spaziali del punto aumentandone o diminuendone la profondità. Questo viene fatto grazie ad un algoritmo che basandosi su una height map, produce un offset alle texture del materiale per accentuare l'effetto di rilievo nella superficie della geometria.[3]

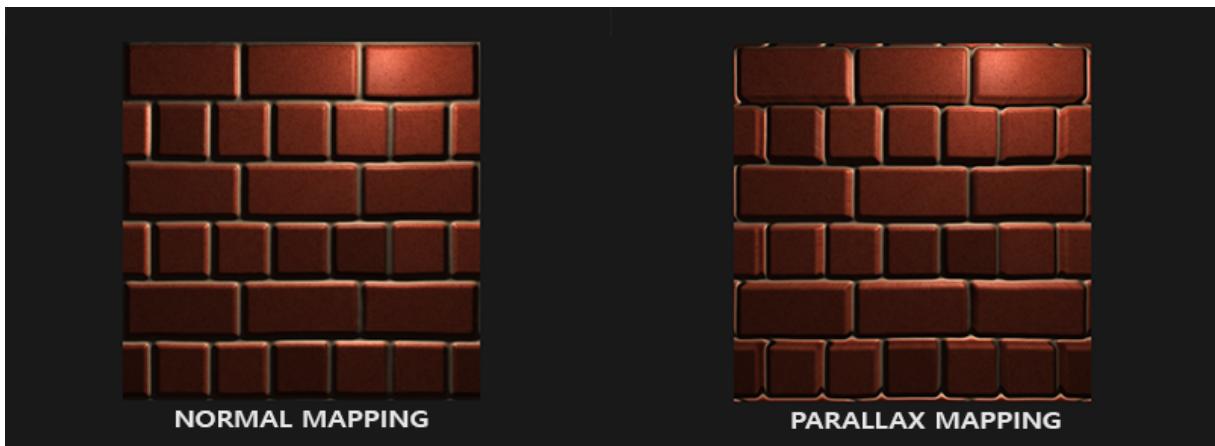


Figura 2.3: Comparazione dell'effetto visivo fra Parallax e Bump mapping.

Infine, WebGi mette a disposizione il Diamond Plugin che renderizza in tempo reale oggetti rifrangenti, in vetro e pietre preziose di alta qualità. La rappresentazione realistica di queste tipologie di materiali risulta sempre molto complessa da realizzare negli altri framework.

2.2.4 Screen Space Post Processing

Il post-processing è il processo di applicazione di filtri ed effetti al buffer di immagini renderizzate prima che vengano visualizzate sullo schermo.[23] Andando a semplificare le sue funzioni: si occupa di modificare una scena già renderizzata, aggiungendo effetti speciali, migliorando le luci, le ombre, i colori e i riflessi per dare all'utente un'immagine più realistica.

WebGi implementa molti effetti di post-processing, il primo di questi è il Progressive Plugin che, sfruttando la progressive rendering pipeline descritta nel core, aggiunge il Super Sample Anti Aliasing (SSAA).

L'aliasing è un effetto visivo che si verifica dopo aver convertito un'immagine reale, in un'immagine in pixel, dopo il processo di quatizzazione che trasforma un segnale continuo, in uno discreto.

Ciò può portare a una rappresentazione distorta dei dettagli dell'immagine e a un effetto di seghettatura, dal momento in cui si vedono le linee dei pixel. Si verifica nella maggior parte delle volte lungo i bordi degli oggetti. Questo effetto è particolarmente fastidioso e rende di bassa qualità la resa grafica

L'SSAAA è un metodo spaziale, che richiede una enorme potenza computazionale, specialmente nella variante non adattiva. L'SSAAA scinde un pixel in più sotto pixel (il numero dipende dal livello del campionamento), effettua una media sul colore, affinché il singolo pixel possa apparire più "omogeneo" una volta che l'SSAAA è stato applicato. Il metodo consiste nel renderizzare l'area di interesse ad una risoluzione più elevata, utilizzando poi i pixel extra per effettuare il calcolo della media e, infine, effettuare il downsampling per ritornare alla risoluzione originale.

Il SSAA applica questo metodo a tutta l'immagine, il che si traduce in una richiesta enorme in termini di memoria grafica e di banda, riducendo le performance. Una versione più evoluta, chiamata SSAA adattivo, applica il tutto ai soli bordi degli oggetti, riducendo

drasticamente la potenza computazionale richiesta.[7]

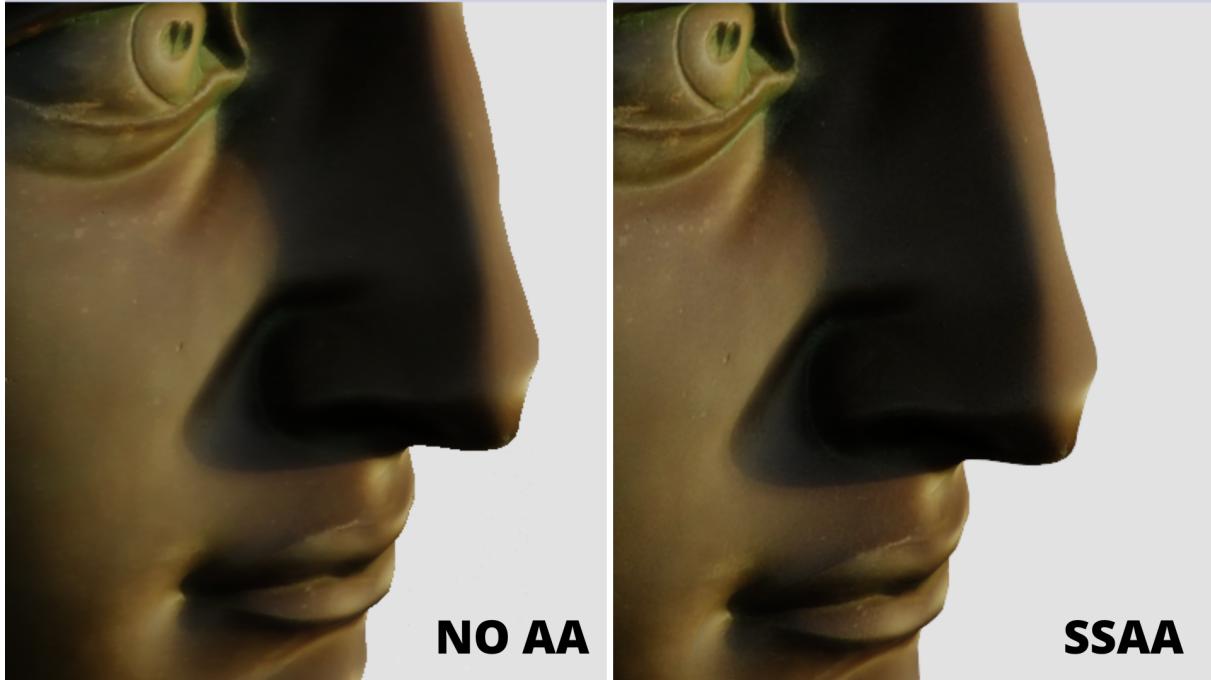


Figura 2.4: Progressive Plugin disattivato vs attivato in WebGi.

In questo caso, si può notare come lungo i bordi del naso nell’immagine NO AA, sia presente il fenomeno dell’aliasing. Questo spiacevole effetto è ridotto e quasi annullato, come si può notare nell’immagine SSAA.

Un altro plugin introdotto è il Tonemapping Plugin. Il tonemapping è una tecnica utilizzata nell’elaborazione delle immagini e nella grafica computerizzata per mappare una serie di colori su un’altra. Questo processo aiuta a compensare la limitata gamma dinamica dello schermo, per evitare la saturazione delle luci alte o l’oscuramento delle luci basse in un’immagine. Così facendo è possibile rappresentare i toni più luminosi di un’immagine sullo schermo.[30]

L’ HDR Bloom Plugin aggiunge l’effetto del Bloom in uno spazio HDR. Il Bloom è l’effetto che si verifica quando un oggetto viene investito da una luce molto intensa e produce un alone luminoso: a causa di effetti di diffrazione questo alone luminoso tende a superare i bordi dell’oggetto stesso.

Lo scopo del Bloom è quello di eliminare o diminuire l’aspetto irreale delle immagini generate al PC.[8] Viene utilizzato molto spesso per migliorare gli effetti luminosi presenti in scene con lampade, neon, fuochi ed altre fonti luminose.



Figura 2.5: Bloom Plugin disattivato vs attivato in WebGi.

Dall’immagine si può notare come le luci del semaforo siano molto più accese e vivaci nell’immagine con il Bloom applicato rispetto quella dove non è stato applicato.

Riguardo la gestione delle ombre, WebGi dispone dello Screen Space Ambient Occlusion (SSAO), utilizzato per realizzare in realtime l’occlusione ambientale nella scena.

L’occlusione ambientale è un effetto visivo che si verifica quando un oggetto si trova dietro ad un altro oggetto e il primo oggetto viene nascosto dal secondo. Nella computer grafica, l’occlusione ambientale viene utilizzata per migliorare il realismo in una scena virtuale, simulando la maniera in cui la luce e l’ombra influiscono sulla percezione degli oggetti.

Le tecniche di occlusione ambientale sono costose perché devono tenere conto della geometria circostante. Si potrebbe sparare un gran numero di raggi per ogni punto nello spazio per determinare la sua quantità di occlusione, ma questo diventa presto computazionalmente inapplicabile per soluzioni in tempo reale. Nel 2007, Crytek ha pubblicato una tecnica chiamata screen-space ambient occlusion (SSAO).

Questa tecnica utilizza il depth-buffer di una scena nello screen space per determinare la quantità di occlusione, invece dei dati geometrici reali durante il rendering. Questo approccio è incredibilmente veloce rispetto all’occlusione ambientale reale e fornisce risultati plausibili.[13]

Sul lato di calcolo, per creare questo effetto, è necessario calcolare la depth map: viene calcolata per ogni pixel della scena, rappresentando la distanza del pixel dalla telecamera. Questa informazione viene utilizzata per determinare la visibilità dei singoli pixel che poi verranno renderizzata dal depth buffer.

Una volta creata la depth map, il SSAO crea un mappa di occlusione ambientale, questa mappa ha al suo interno tutti pixel della scena e tiene traccia di quanto ognuno sia occluso dalla luce ambientale.

Successivamente viene applicato il blending della scena, ovvero vengono mescolati i pixel della mappa di occlusione ambientale e quelli della scena originale.[31]



Figura 2.6: SSAO Plugin disattivato vs attivato in WebGi.

In questo caso, si può apprezzare come con il SSAO Plugin attivo, nella zona dei tasti della tastiera siano presenti molte ombre dovute all'occlusione ambientale.

Uno degli effetti di post-processing più difficili da implementare è lo Screen Space Reflections (SSR), WebGi lo realizza e viene usato per calcolare realtime le riflessioni all'interno della scena.

L'SSR viene quindi utilizzata molto spesso per realizzare riflessioni realistiche su superfici riflettenti come acqua o specchi. Invece di calcolare i riflessi utilizzando informazioni sulla geometria e la luce nella scena, SSR utilizza informazioni sulle immagini già disegnate

sullo screen space.

Il funzionamento di base della SSR consiste nel cercare di identificare le superfici riflettenti e di mappare le riflessioni sulle immagini già disegnate.

Il principio di funzionamento della SSR inizia dal rilevamento delle superfici riflettenti. Questo viene solitamente fatto utilizzando informazioni sulla profondità e sulla normale della superficie. Infatti, Le superfici che hanno una normale rivolta verso la fonte di luce e che hanno una certa riflettività sono considerate come superfici riflettenti.

Dopo avviene il tracing dei raggi: una volta rilevate le superfici riflettenti, vengono tracciati dei raggi dalla superficie riflettente verso la fonte di luce per cercare corrispondenze con altre parti dell'immagine. Questo può essere fatto utilizzando una depth map, che, come descritto in precedenza nella SSAO, rappresenta la distanza di ogni punto della scena dalla telecamera.

Infine, tracciati i raggi, vengono utilizzate le informazioni ottenute per mappare le riflessioni sulle superfici riflettenti. Generalmente viene fatto utilizzando una texture o una mappa delle riflessioni pre-calcolate. In questo modo, le riflessioni possono essere create in modo efficiente e realistico senza dover ricalcolare tutta la scena.[14]

È importante notare che la SSR ha delle limitazioni rispetto a tecniche più avanzate come la Global Illumination. Ad esempio, SSR non tiene conto delle interazioni tra le luci e le superfici nella scena, quindi le riflessioni potrebbero non essere sempre corrette o realistiche. WebGi ha in fase di dev questo plugin, è possibile utilizzarlo, ma presenta ancora alcune imperfezioni.



Figura 2.7: SSR Plugin disattivato vs attivato in WebGi.

Dalla figura si nota come, con il plugin attivo, lo specchio riflette le mesh circostanti e cambia leggermente il colore di alcuni materiali, dal momento il cui avranno dei riflessi che nell'immagine di sinistra, senza SSR, non sono presenti.

Per contrastare l'aliasing, WebGi decide di implementare il Temporal Anti Aliasing (TAA) Plugin. L'altro metodo di anti-alising sviluppato: il SSAA, mira ad eliminare principalmente l'aliasing su immagini statiche.

Il TAA è una tecnica avanzata di anti-aliasing che viene utilizzata per ridurre gli effetti di sgranatura su immagini in movimento, ovvero quando si muove la camera.

Il TAA si basa sull'idea di campionare l'immagine non solo in un singolo momento, ma anche in diversi momenti del tempo, facendo quindi un campionamento temporale. In pratica, il TAA utilizza più immagini generate in momenti successivi e le combina insieme in modo da ottenere un'immagine finale con un aspetto più uniforme e privo di aliasing.

Il processo di TAA prevede di salvare in un History Buffer i frame per poi unirli per ottenere l'immagine finale. Al campionamento dell'immagine viene applicato il jittering, che introduce una leggera variazione nella posizione dei campioni dell'immagine, in modo che ogni immagine nell'History Buffer sia leggermente diversa dall'altra. Questo aiuta a ridurre l'effetto di flickering, ovvero l'aspetto come lampeggiante che può verificarsi

quando le immagini vengono combinate insieme.

Inoltre, il jittering aiuta a ridurre l'effetto di ghosting, ovvero il fenomeno che può verificarsi quando oggetti in movimento appaiono sfocati o doppi nell'immagine. Introducendo una leggera variazione nella posizione dei campioni, il jittering rende più difficile per il cervello percepire le differenze tra i frame, riducendo l'effetto di ghosting.[11]

Ciò significa che il TAA è in grado di fornire una qualità visiva migliore rispetto ad altre tecniche di anti-aliasing come MSAA (Multi-Sample Anti-Aliasing) o SSAA, che non tengono conto dei movimenti degli oggetti.

Tuttavia, il TAA ha anche alcuni svantaggi. Ad esempio, richiede una quantità maggiore di calcolo rispetto ad altre tecniche di anti-aliasing.

Nonostante ciò, il TAA è una tecnica molto popolare in giochi e video in tempo reale, e viene spesso utilizzata insieme ad altre tecniche di anti-aliasing per ottenere una qualità visiva ancora migliore. WebGi infatti mira ad eliminare l'aliasing a livello statico attivando il SSAA e quello in movimento attivando il TAA, sfruttando al meglio le potenzialità dei due algoritmi di post-processing.[7][33]

2.2.5 Luci e setup della scena

Riguardo il settaggio delle luci e il setup della scena WebGi fornisce alcuni plugin molto interessanti. Il primo di questi è il Ground Plugin: inserisce automaticamente il pavimento nella scena, con la possibilità di aggiungervi le ombre baked, screen space reflections, il background blending ed altre opzioni.

Le ombre baked sono un tipo di ombre pre-calcolate che vengono generate in anticipo e memorizzate nella texture della scena, anziché essere calcolate in realtime durante la renderizzazione. Questo tipo di ombre viene utilizzato per ottimizzare le prestazioni e migliorare la qualità visiva delle ombre nella scena.

Le screen space reflections del pavimento vengono gestite in maniera analoga all'SSR Plugin.

Il Background Blending è una tecnica di computer grafica che consente di sovrapporre un'immagine o un elemento grafico su un'altra immagine o sfondo, creando un effetto di trasparenza o di sovrapposizione.

Questo effetto viene utilizzato in combinazione con un'immagine HDR, infatti, quando si utilizza un HDR con il Background Blending, le immagini HDR vengono utilizzate per creare un'immagine composta che combina elementi da due o più immagini diverse. Questo permette di ottenere un'immagine più dettagliata e realistica, poiché la tecnologia HDR consente di rappresentare una gamma più ampia di luminosità e colori.

Questo può aiutare a creare immagini più convincenti e realistiche migliorando la gamma delle luci e colori e delle ombre. Di default WebGi unisce un'immagine HDR di base, combinata ad uno sfondo con colore di codifica `rgb fefefe`.

Nell'immagine che segue si può notare come, nella figura con il Ground Plugin attivo, si noti un pavimento riflettente e con le ombre, rispetto alla figura dove è disattivato, che render il soggetto come "per aria".



Figura 2.8: Ground Plugin disattivato vs attivato in WebGi.

2.2.6 Animazioni

Per la realizzazione delle animazioni WebGi disponi di tre plugin differenti.

Il primo di questi è il GLTF Animation Plugin, che consente di creare e gestire animazioni 3D utilizzando il formato GLTF (GL Transmission Format).

GLTF è un formato aperto per il trasferimento e l'archiviazione di modelli 3D e le relative

animazioni, sviluppato per essere utilizzato con tecnologie web come WebGL.

Il GLTF Animation Plugin consente di importare animazioni 3D in formato GLTF. Questo può essere utile per i designer 3D che desiderano utilizzare animazioni 3D nelle loro applicazioni web o in altri software. Inoltre, il formato GLTF supporta animazioni ad alta fedeltà e ha una piccola dimensione dei file, il che lo rende ideale per la trasmissione e il rendering di animazioni 3D su Internet.

Si presta compatibile ad animazioni ideate in Blender, Maya e 3ds Max, per cui l'utente, dopo aver creato l'animazione sul software di animazione, deve solamente gestirne i suoi stadi all'interno di WebGi, attraverso i comandi di pause, resume e stop, oppure attraverso playClip o playClips che richiede il nome dell'animazione o delle animazioni da eseguire.

```

1 const gltfAnims = viewer.getPlugin(GLTFAnimationPlugin);
2
3 // Play all animations
4 gltfAnims.playAnimation()
5 // Pause animation
6 gltfAnims.pauseAnimation()
7 // Resume animation
8 gltfAnims.playAnimation()
9 // Stop animation
10 await timeout(3000) // 3 secs
11 const reset = true // optionally reset the animation to time=0
12 gltfAnims.stopAnimation(reset)
13
14 const resetOnEnd = true // Optionally, reset the animation to time=0 after end.
15 await gltfAnims.playClip('Cube Anim', resetOnEnd) // Pass the name of the animation clip.
16 console.log("Animation ended")
17
18 // or play multiple clips
19 await gltfAnims.playClips(['Cube Anim', 'Sphere Anim', resetOnEnd]) // This will return
    when all animations are finished.

```

Listing 2.5: Esempio base di utilizzo del GLTF Animation Plugin.

Se si vuole rendere la scena interattiva da WebGi, senza utilizzare animazioni esterne, è possibile utilizzare il Camera View Plugin, che permette di salvare punti di vista e di creare transizioni fra essi.

Il suo funzionamento si basa sul salvare la posizione della camera ed applicarle una trasformazione 3D di rotazione e traslazione per portarla alla posizione successiva.

L'ultimo plugin messo a disposizione da WebGi è il Popmotion Plugin, esso permette di

creare il tweening tra due frame e di visualizzare animazioni. Si basa su popmotion.io, una libreria funzionale scritta in JavaScript dedicata alla gestione delle animazioni.

Il tweening delle animazioni è una tecnica di animazione che consiste nell'interpolare il movimento di un oggetto in modo graduale tra due o più keyframe, detti anche posizioni chiave. Il processo di tweening prevede la definizione di un insieme di proprietà dell'oggetto da animare, come la posizione, la rotazione o la scala, per ogni frame dell'animazione.[32] Popmotion quindi calcola le posizioni intermedie fra i keyframe per rendere le animazioni più fluide.

2.2.7 Export e render

L'editor online è un'applicazione online accessibile direttamente dal sito di WebGi. Consente di importare i propri modelli 3D all'interno di un canvas e, tramite l'interfaccia grafica, è possibile modificare i parametri dei plugin, con la possibilità anche di attivarli e disattivarli. Questa feature non è sostitutiva dei software di modellazione, infatti si può modificare la posizione, la scala, la rotazione, le proprietà dei materiali, ma non si può modellare.

Il suo utilizzo può risultare molto interessante dal momento in cui si vuole visualizzare in realtime le modifiche che vengono fatte alla scena con la gestione dei plugin. Può essere visto quindi solo come un creatore e un compositore di scene.

Inoltre, una volta creata la scena, WebGi implementa un plugin per l'esportazione e il render di immagini o video.

L'Asset Exporter Plugin permette di esportare modelli in formato GLB oppure degli asset preset, che possono essere riutilizzati nella creazione di altre scene.

La scena creata attraverso l'editor può quindi essere esportata, per ora solamente in formato GLB. Il vantaggio di utilizzare questo processo è che nel modello glb esportato è possibile scegliere quali plugin esportare e anche i loro preset. In questo modo, tornando al codice, si può inserire il file in formato .glb attraverso l'asset manager e la scena sarà la stessa che si è creata online.



Figura 2.9: Editor online in WebGi.

Nella fase di esportazioni può essere utilizzato il GLTF Draco Export Plugin che aggiunge un supporto per l'applicazione della compressione Draco al modello dopo la fase di export.

Draco è una libreria per la compressione e la decompressione di mesh geometriche 3D e nuvole di punti sviluppata da Google. Ha lo scopo di migliorare l'archiviazione e la trasmissione di grafica 3D.

Draco è stato progettato e realizzato per garantire efficienza e velocità di compressione. Il codice supporta la compressione di punti, informazioni di connettività, coordinate di texture, informazioni di colore, normali e qualsiasi altro attributo generico associato alla geometria. Draco viene rilasciato come codice sorgente C++ che può essere utilizzato per comprimere la grafica 3D e come decodificatore C++ e Javascript per i dati codificati.[9]

Per il render di immagini e video si possono utilizzare rispettivamente il Canvas Snipper Plugin e il Canvas Recorder Plugin.

Il primo consente di creare un'immagine e di effettuare il download: alcune delle immagini presenti in questo documento sono state create attraverso questo plugin; il secondo, invece, renderizza un video in formato .webm o una sequenza di immagini.

2.2.8 Plugin di supporto

L’interazione con l’utente è fondamentale all’interno di una scena 3D, ma soprattutto deve risultare intuitivo anche utilizzare un certo framework, attraverso il Dropzone Plugin è possibile trascinare modelli 3D, nel formato necessario, direttamente all’interno dell’editor online. Questa funzionalità risulta essere molto comoda, poiché, una volta trascinato il modello, è possibile modificarlo a piacere e poi esportarlo con tutte le caratteristiche aggiunte e in formato ottimizzato.

Un’altra funzionalità del Dropzone Plugin è la possibilità di utilizzare l’auto center e l’auto scale. Il modello 3D viene automaticamente scalato e posto al centro della scena, ciò risulta particolarmente comodo ed utile qualora si realizzino scene con un singolo oggetto.

Un altro plugin molto interessante è il Picking Plugin che abilita le interazioni e l’interfaccia utente per selezionare un oggetto per poterne inoltre modificarne le proprietà.

Il picking può essere gestito anche attraverso gli eventi del mouse differenziando la selezione se si è premuto sopra una mesh col puntatore oppure se si è solamente sopra.

```
1 const picking = viewer.addPlugin(PickingPlugin);
2 picking.addEventListener('hitObject', (e)=>{
3     console.log('Hit Event:', e);
4     console.log('Hit Time:', e.time);
5     console.log('Hit Intersect:', e.intersects.intersect);
6     console.log('Hit Object:', e.intersects.selectedObject);
7
8     // To disable object selection, set 'e.intersects.selectedObject' to null before
9     // returning
10    e.intersects.selectedObject = null
11    // or change the selection to something else
12    e.intersects.selectedObject = e.intersects.selectedObject.parent
13})
```

Listing 2.6: Esempio base di utilizzo del Picking Plugin.

Nello sviluppo del progetto si rivelerà un componente fondamentale nella fase di sviluppo. Inoltre, WebGi implementa un plugin utile per modificare i materiali di un modello: il Material Configurator Plugin.

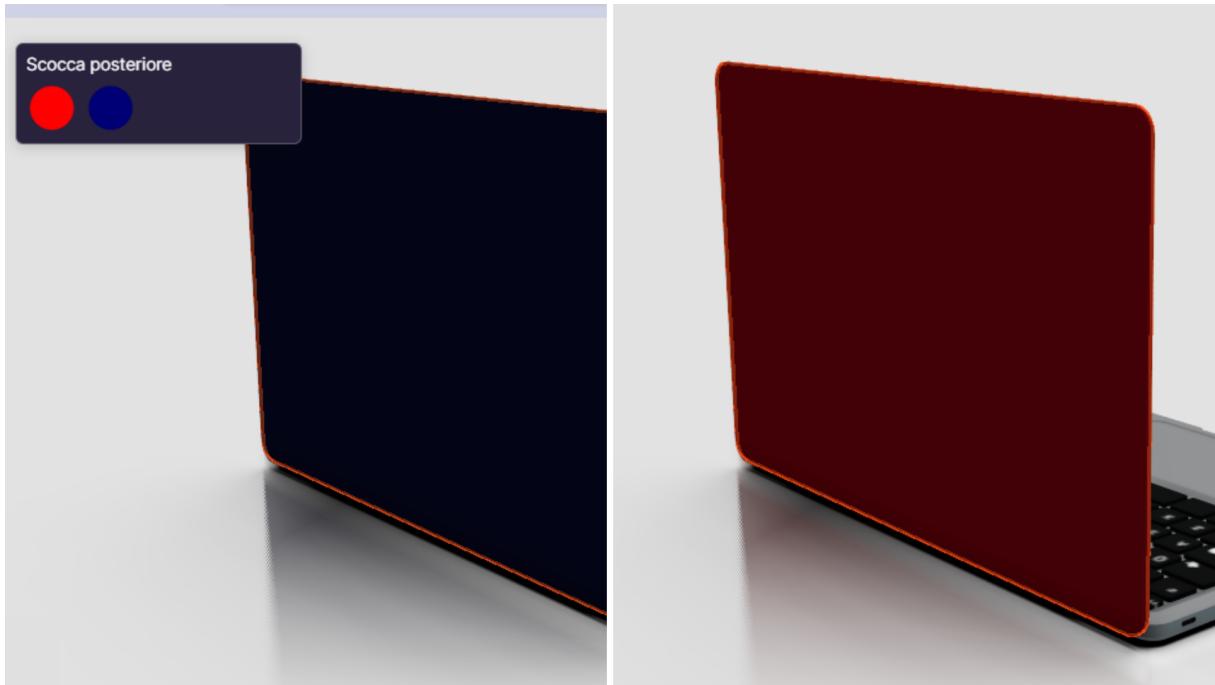


Figura 2.10: Material configurator nell’editor online di WebGi.

Si occupa della gestione della configurazione dei materiali, può essere integrato ad una interfaccia UI scritta in js.

```

1 class CustomMaterialConfiguratorPlugin extends MaterialConfiguratorBasePlugin {
2
3     // This must be set to exactly this.
4     public static PluginType = 'MaterialConfiguratorPlugin'
5
6     // this function is automatically called when an object is loaded with some material
7     // variations
8     protected async _refreshUi(): Promise<boolean> {
9         if (!await super._refreshUi()) return false // check if any data is changed.
10
11         for (const variation of this.variations) {
12             console.log(variation.title)
13
14             variation.materials.map(material => {
15                 // material is the variation that can be applied to an object
16
17                 let image: any
18                 if (!variation.preview.startsWith('generate:')) {
19                     const pp = (material as any)[variation.preview] || '#ff00ff'
20                     image = pp.image || pp
21                 } else {
22                     // Generate a small snapshot of the material preview based on some shape (optional)
23                     image = this._previewGenerator!.generate(material, variation.preview.split(':')[1] as
24                         any)
25                 }
26             })
27         }
28     }
29 }
```

```

23 }
24 // callback to change the material variations
25 const onClick = ()=>{
26   this.applyVariation(variation, material.uuid)
27 }
28 // Generate a UI from this data.
29 console.log({uid: material.uuid, color: (material as any).color, material: material,
30             image, onClick})
31 }) }
32 return true
33 }
```

Listing 2.7: Sviluppo di una UI per la configurazione dei materiali.

2.2.9 Plugin e ShaderMaterial personalizzati

Oltre i numerosi plugin realizzati da WebGi, c’è la possibilità di scrivere plugin personalizzati. Se si vuole creare un plugin si deve implementare un IVIEWERPLUGIN per essere collegati al visualizzatore.

```

1 interface IVIEWERPLUGIN extends IEventDispatcher<string>, IUICONTAINER,
2   Partial<IJSONSERIALIZABLE> {
3     // all classes must have this static property with a unique identifier value for this
4     // plugin
5     static readonly PLUGINTYPE: string
6
7     // these plugins will be added automatically(with default settings), if they are not
8     // added yet.
9     dependencies?: Class<IVIEWERPLUGIN<any>>[]
10
11    // the viewer will render the next frame if this is set to true
12    dirty?: boolean;
13
14    // Called when this plug-in is added to the viewer
15    onAdded(viewer: TVIEWER): Promise<void>;
16
17    // Called when this plug-in is removed from the viewer
18    onRemove(viewer: TVIEWER): Promise<void>;
19 }
```

Listing 2.8: Interfaccia IVIEWERPLUGIN in WebGi.

Volendo, per semplificare ed eliminare il boilerplate, si possono usare le classi astratte AViewerPlugin, GenericFilterPlugin, MultiFilterPlugin. Il boilerplate è un codice preimpostato che funge da base per poterne implementare uno personalizzato, come in questo caso.

WebGi fornisce un esempio di plugin personalizzato.

```

1 import {AViewerPlugin, IEvent, reflHelpers, serialize, ViewerApp, uiFolder, uiToggle,
2         uiSlider, uiInput, uiButton, onChange} from 'webgi'
3
4 export class SamplePlugin extends AViewerPlugin<'sample-1'|'sample-2'> { // These are
5     the list of events that this plugin can dispatch.
6
7     static readonly PluginType = 'SamplePlugin' // This is required for serialization
8         and handling plugins. Also used in viewer.getPluginByType()
9
10    @serialize() // Adds this property to the list of serializable. This is also used
11        when serializing to glb in AssetExporter.
12    enabled = true
13
14    // A plugin can have custom properties.
15    @serialize('someNumber')
16    @onChange(SamplePlugin.prototype._updateParams) // this function will be called
17        whenever this value changes.
18    val1 = 0
19
20    // A plugin can have custom properties.
21    @onChange(SamplePlugin.prototype._updateParams) // this function will be called
22        whenever this value changes.
23    @serialize()
24    val2 = 'Hello'
25
26    public printValues = () => {
27        console.log(this.val1, this.val2)
28        this.dispatchEvent({type: 'sample-1', detail: {sample: this.val1}}) // This will
29            dispatch an event.
30    }
31
32    constructor() {
33        super()
34        this._updateParams = this._updateParams.bind(this)
35    }
36
37    private _updateParams() {
38        console.log('Parameters updated.')
39        this.dispatchEvent({type: 'sample-2'}) // This will dispatch an event.
40    }
41
42 }
```

```

34     async onAdded(v: ViewerApp): Promise<void> {
35         await super.onAdded(v)
36
37         // Do some initialization here.
38         this.val1 = 0
39         this.val2 = 'Hello'
40
41         v.addEventListener('preRender', this._preRender)
42         v.addEventListener('postRender', this._postRender)
43         v.addEventListener('preFrame', this._preFrame)
44         v.addEventListener('postFrame', this._postFrame)
45
46         this._viewer!.scene.addEventListener('addSceneObject', this._objectAdded) //  

47             this._viewer can also be used while this plugin is attached.
48     }
49
50
51     async onRemove(v: ViewerApp): Promise<void> {
52         // remove dispose objects
53
54         v.removeEventListener('preRender', this._preRender)
55         v.removeEventListener('postRender', this._postRender)
56         v.removeEventListener('preFrame', this._preFrame)
57         v.removeEventListener('postFrame', this._postFrame)
58
59         this._viewer!.scene.removeEventListener('addSceneObject', this._objectAdded) //  

60             this._viewer can also be used while this plugin is attached.
61
62         return super.onRemove(v)
63     }
64
65     // async onDispose(viewer: ViewerApp): Promise<void> {
66     //     // this is optional
67     //     return super.onDispose(viewer);
68     // }
69
70     private _objectAdded = (ev: IEvent<any>)=>{
71         console.log('A new object, texture or material is added to the scene.',  

72             ev.object)
73     }
74     private _preFrame = (ev: IEvent<any>)=>{
75         // This function will be called before each frame. This is called even if the  

76             viewer is not dirty, so it's a good place to do viewer.setDirty()
77     }
78     private _preRender = (ev: IEvent<any>)=>{
79         // This is called before each frame is rendered, only when the viewer is dirty.

```

```

78     }
79     // postFrame and postRender work the same way as preFrame and preRender.
80
81 }
```

Listing 2.9: Esempio base di plugin personalizzato fornito da WebGi.

Inoltre, nella pagina ufficiale, è possibile notare il codice comprensivo della UI che permette di cambiare i possibili parametri del plugin.

Oltre questo esempio, ne viene fornito uno che utilizza lo ShaderPass. Lo Shader Pass è un fase di elaborazione delle informazioni di un oggetto o di un'immagine durante il processo di rendering. In ogni pass, uno shader esegue specifiche operazioni matematiche sulle informazioni dell'immagine, come ad esempio la manipolazione della luce, l'applicazione di texture o la creazione di effetti speciali. Una volta che un pass è completato, le informazioni vengono inviate alla pipeline di rendering per il successivo pass fino a quando non viene ottenuto l'output finale.[26]

2.2.10 Esempio di creazione di una scena

Dopo aver analizzato le varie funzionalità dei plugin, è necessario descrivere il processo di creazione di una scena virtuale in WebGi.

Per comodità è consigliato di usare il template da loro proposto, andando a clonare la repository presente su GitHub. Dopo aver installato le dipendenze con il comando *npm install* da terminale, è già possibile lanciare, attraverso il comando *npm start*, l'applicazione web.

Il template comprende tutto il necessario per creare una scena base, tra cui un modello 3D. Analizzando più nel dettaglio quanto presente, si può notare che all'interno del file index.html.

```

1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4   <title>WebGi starter</title>
5   <meta charset="UTF-8" />
6   <meta name="viewport" content="width=device-width, user-scalable=no,
      minimum-scale=1.0, maximum-scale=1.0">
7 </head>
8 <body>
```

2.2 – LA STRUTTURA DI WEBGI

```
9 <div id="webgi-canvas-container">
10   <canvas id="webgi-canvas"></canvas>
11 </div>
12
13 <script src="src/index.ts">
14 </script>
15 </body>
16 </html>
```

Listing 2.10: File .html presente nel template di WebGi.

L’elemento canvas webgi-canvas è il più importante da tenere in considerazione poiché, all’interno del file index.ts verrà inizializzato come canvas del viewer 3D. Passando al file index.ts è presente uno scheletro per la creazione di una scena, dove è presente solo il viewer e l’asset manager per caricare un modello già presente nel template.

```
1 <import {
2   ViewerApp ,
3   AssetManagerPlugin ,
4   TonemapPlugin ,
5
6   addBasePlugins ,
7   ITexture , TweakpaneUiPlugin , AssetManagerBasicPopupPlugin , CanvasSnipperPlugin ,
8
9   IViewerPlugin ,
10 } from "webgi";
11 import "./styles.css";
12
13 async function setupViewer(){
14
15   // Initialize the viewer
16   const viewer = new ViewerApp({
17     canvas: document.getElementById('webgi-canvas') as HTMLCanvasElement ,
18   })
19
20   // Add some plugins
21   const manager = await viewer.addPlugin(AssetManagerPlugin)
22
23   // Add a popup(in HTML) with download progress when any asset is downloading.
24   await viewer.addPlugin(AssetManagerBasicPopupPlugin)
25
26   // or use this to add all main ones at once.
27   await addBasePlugins(viewer)
28
29   // This must be called once after all plugins are added.
30   viewer.renderer.refreshPipeline()
31 }
```

```

32     await manager.addFromPath("./assets/classic-watch.glb")
33
34     const uiPlugin = await viewer.addPlugin(TweakpaneUiPlugin)
35     uiPlugin.setupPlugins<IViewerPlugin>(TonemapPlugin, CanvasSnipperPlugin)
36
37 }
38
39 setupViewer()

```

Listing 2.11: File index.ts presente nel template di WebGi.

Dopo aver importato i plugin necessari viene istanziata la funzione *setupViewer* dove, in primo luogo, viene inizializzato il viewer, andando a selezionare come canvas l'elemento HTML denominato "webgi-canvas", successivamente viene istanziato il manager tramite l'AssetManagerPlugin ed aggiunti alcuni dei plugin di base. Attraverso la funzione del viewer renderer *refreshPipeline()* viene aggiornata la pipeline di rendering in base ai plugin che sono stati aggiunti. Successivamente, viene aggiunto il modello 3D alla scena tramite il manager e, in questo caso, il plugin relativo alla UI.

Attraverso il comando *npm start* viene fatta la build e da locale si può visualizzare la scena ed interagire con il modello. Essendo lo scheletro di una scena in WebGi, non sono presenti i Plugin, sta quindi allo sviluppatore capire quali sono necessari ed andare ad importarli ed utilizzarli. Questa è la webapplication che è creata dal template

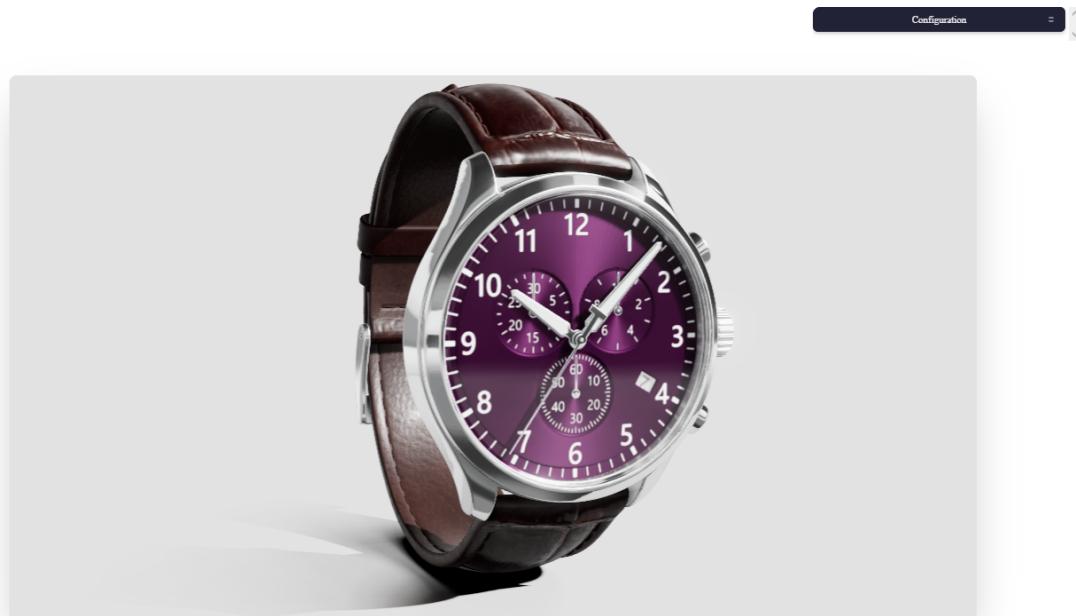


Figura 2.11: Scena prodotta del tamplate di WebGi.

2.2.11 Feature in sviluppo e limiti

Tra i plugin all'interno di WebGi, ce ne sono alcuni in fase di sviluppo che nel futuro prossimo potrebbero sicuramente migliorare l'esperienza visiva e di utilizzo dell'utente.

Il Path Tracing Plugin simula il ray-tracing nello shader utilizzando un BVH tree. Il Ray Tracing è una tecnica di rendering capace di produrre effetti di luce, specialmente riflessi, incredibilmente realistici, viene utilizzato nell'industria cinematografica, ma nello sviluppo di scene 3D in real time, devono accedere ad un grande potenza computazionale.

Il ray-tracing simula l'illuminazione del mondo reale. Consiste nel puntare una luce su un oggetto e tracciare come la luce rimbalza sulla sua superficie. Va esteso questo concetto a molti raggi, sfruttando quelli che tornano (e non tornano) per capire come dovrebbe apparire la scena. Ad esempio, i raggi che non sono tornati sono stati probabilmente bloccati da un oggetto, creando così un'ombra, proprio come avviene per i radar.

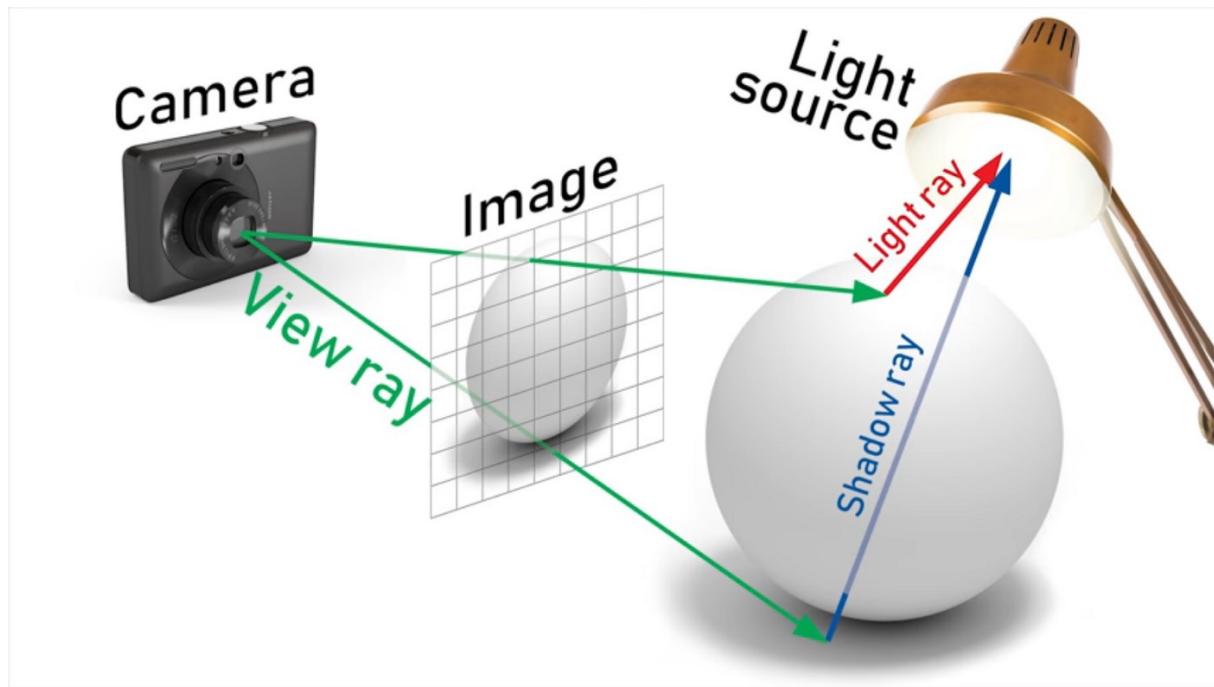


Figura 2.12: Funzionamento del ray-tracing.

Il Ray Tracing è esattamente questo, una tecnologia che consente ombre e riflessi notevolmente più realistici, insieme a translucenza e dispersione notevolmente migliorate.

L'algoritmo tiene conto del punto in cui la luce colpisce un oggetto e calcola l'interazione proprio come l'occhio umano la elaborerebbe, influenzando anche i colori che si vedono.

La quantità di calcolo necessaria perché l'algoritmo tracci il percorso della luce e simulare il modo in cui questa interagisce con gli oggetti virtuali che colpisce nel mondo generato al computer, è enorme.[18]

L'utilizzo del BVH tree (Boundary Volume Hierarchy Tree) permette l'accelerazione della computazione dell'intersezione dei raggi. Infatti, un BVH tree è una struttura di dati utilizzata per organizzare e rappresentare gli oggetti 3D in una scena in modo efficiente, rendendo più veloci le operazioni di rendering e collision detection.

Il BVH Tree suddivide la scena in regioni che contengono i singoli oggetti oppure gruppi di oggetti, e utilizza un albero gerarchico per organizzare queste regioni in maniera efficiente.

Per l'implementazione WebGi sfrutta un gpu-path-tracer sviluppato per three.js e la repository è consultabile liberamente.

In three.js è possibile sfruttare la fisica attraverso i moduli cannon.js e ammo.js, in WebGi è in fase di sviluppo la creazione dei relativi plugin.

Ammo.js è una libreria javascript basata sulla tecnologia Bullet Physics, uno dei motori di simulazione fisica più conosciuti. Viene utilizzato per simulare la fisica dei corpi rigidi e dei materiali deformabili come la pelle, la gelatina e il tessuto.

Cannon.js è una libreria JavaScript per la realizzazione realistica della fisica relativa alla dinamica dei corpi rigidi e della collision detection.

Infine, è in fase di sviluppo il plugin relativo alla realtà aumentata, che permetterebbe di visualizzare i modelli 3D attraverso il proprio cellulare tramite la fotocamera oppure attraverso un visore ottico.

Il limite principale riscontrato però, è il prezzo. A livello aziendale è stato chiesto il costo della licenza per poter utilizzare WebGi per scopi commerciali e le somme proposte risultano disallineate con quelle del mercato dei software per il rendering realtime di scene 3D.

Capitolo 3

Sviluppo del Progetto

All'interno di questo capitolo viene descritto, in base ai requisiti posti dall'azienda, quello che è il lavoro svolto per creare un'applicazione web con WebGi.

3.1 Analisi dei requisiti

Il primo requisito funzionale richiesto è quello di sviluppare un'applicazione web con WebGi, mantenendo tutte le funzionalità ben sviluppate nel precedente software: picking degli oggetti, la struttura di movimento orbitale della camera e la modalità di focus su un singolo prodotto. In aggiunta, oltre a mantenere queste funzionalità è stato richiesto di inserire le animazioni ai prodotti e una schermata di caricamento durante il loading della scena.

Quindi, è necessario, anche sfruttando il metodo di interazione del software sviluppato in precedenza, adattare alcune funzionalità, trasformando la loro implementazione con l'uso dei plugin. Il fine è quello di comprendere le potenzialità di WebGi e, posto a confronto con il software precedente, quali sono le differenze in termini di performance e di resa grafica.

Inoltre, è stata richiesta l'aggiunta di un'interfaccia grafica, detta "infografica", dei prodotti quando vengono visualizzati nel dettaglio. Devono essere mostrati: il nome del prodotto, la descrizione, un bottone che rimandi allo shop online del venditore e bottoni play/pause/stop di gestione delle animazioni.

I requisiti non funzionali invece, sono: migliorare la resa grafica dei materiali, migliorare le performance in termini di fps e carico computazionale, migliorare la fluidità di interazione del mouse, ridurre il tempo di risposta e ridurre il tempo di caricamento della scena.

Infine, un altro requisito funzionale, ma di secondaria importanza è quello di capire se è possibile realizzare un workflow con WebGi che possa facilmente e velocemente sviluppare le scene virtuali.

Per cui l’obiettivo è far sì che, dalla creazione del modello, siano necessari pochi e semplici passaggi per trasformarlo da un modello 3D ad una scena virtuale, senza dover modificare in modo sostanziale il codice.

3.2 Analisi del problema

Il problema di mantenere alcuni dei servizi del vecchio software, ma aggiungendo animazioni e una scena di caricamento della scena virtuale richiede una strategia di integrazione efficace. Ciò può comportare l’utilizzo di librerie di animazione e grafica esistenti, la definizione di standard per la creazione di nuove animazioni, l’aggiunta di funzionalità di caricamento personalizzate e la creazione di un’interfaccia utente intuitiva per l’utilizzo di queste funzionalità.

Riguardo l’integrazione di librerie, WebGi fornisce già un plugin dedicato alle animazioni e di conseguenza è necessario integrare solamente un’interfaccia utente dedicata alla loro gestione. Per la schermata di caricamento è possibile sfruttare gsap: una delle librerie JavaScript di animazione più performante ed utilizzate che ha anche un’integrazione con l’elemento HTML canvas.

Il problema di sviluppare un’infografica che mostri nel dettaglio il prodotto selezionato richiede una conoscenza approfondita del prodotto e del pubblico a cui è destinata l’infografica. Ciò potrebbe comportare la creazione di un database di informazioni sulla caratteristiche del prodotto, la definizione di un layout grafico e un’organizzazione appropriata delle informazioni. Grazie alle conoscenze pregresse, si è propeso a sviluppare l’infografica attraverso componenti HTML aggiungendoci lo stile in CSS. Riguardo il popolamento della grafica con le informazioni del prodotto è stato deciso, per il poco tempo

a disposizione, di utilizzare un oggetto JSON che contenesse tutte le informazioni necessarie, utilizzando come identificativo il nome del gruppo del prodotto, in modo da poterli accoppiare al meglio.

Il problema di migliorare le prestazioni in termini di fps e CPU, migliorare la fluidità di movimento del mouse e ridurre il tempo di caricamento della scena richiede una valutazione attenta dell’architettura software esistente e l’identificazione di aree in cui è possibile ottimizzare il codice e l’utilizzo delle risorse di sistema. Ciò potrebbe comportare la riduzione della complessità della scena, l’ottimizzazione del rendering, l’uso di tecniche di caching per migliorare la velocità di caricamento e l’ottimizzazione del codice per ridurre il carico sulla CPU.

In questo caso WebGi opera già molte di queste migliorie, andando a modificare il carico di stress della CPU e della GPU in base alla necessità e migliorando dove possibile il rendering attraverso algoritmi di post processing e applicando materiali realistici. Riguardo l’architettura del codice si è deciso di creare un nuovo file: *util.ts* che contenesse solamente le funzioni secondarie, lasciando il file *index.ts* come core contenente solo le funzioni principali. Riguardo la velocità di caricamento, molto dipende dalla dimensione del modello, è possibile applicare compressioni, come la DRACO compression, per ridurne le dimensioni. Lo sviluppo di mobili per il makeup, contenenti molti prodotti e dettagli, porta il modello a pesare dai 50MB ai 60MB. Questa mole di dati è necessario che venga comunque scaricata la prima volta che si apre la scena su un nuovo dispositivo. Nelle volte successive, attraverso il caching, il tempo di caricamento è drasticamente ridotto. In termini di ottimizzazione del software tenere separate le funzionalità core e quelle secondarie permette di aumentare leggermente le prestazioni. Il problema di migliorare la pipeline di sviluppo di una scena virtuale, rendendola il più semplice possibile, richiede una valutazione attenta dei processi atti alla creazione e l’identificazione di aree in cui è possibile semplificare e ottimizzare il flusso di lavoro. Ciò potrebbe comportare l’introduzione di nuovi strumenti o tecnologie, la riduzione del numero di passaggi necessari per completare una determinata attività e la standardizzazione di procedure e modelli.

In tal caso, attraverso l’utilizzo di un nuovo framework, è possibile ridurre il numero di passaggi necessari alla creazione del modello 3D, andando a rimuovere molte delle problematiche sulla creazione del modello, soprattutto nella creazione dei placeholder e dei

template. In tal caso, sarebbe utile sviluppare una pipeline che si occupi solo di importare il nuovo modello 3D nella scena e cambiare solamente il path nel plugin dell’asset manager. Ma lo sviluppo di un nuovo workflow che automatizzi la creazione di una scena è realizzabile solamente se non si vogliono porre modifiche al codice, banalmente, modificare le impostazioni di un plugin, può portare ad una modifica sostanziale del sistema.

3.3 Sviluppo del progetto

La struttura del progetto si divide in due file principali *index.ts* e *utils.ts*, nel primo viene definito il core dell’applicazione web, con l’aggiunta dei plugin necessari, nel secondo sono definite tutte le funzioni di supporto.

Per creare una scena è prima necessario creare il modello in Blender, per semplificare lo di sviluppo della scena è stato modificato il metodo di nomenclatura delle mesh e dei materiali del modello 3D, rendendo il processo di creazione il più veloce possibile.

I prodotti all’interno del mobile avranno lo stesso nome degli empty, affiancati da un numero progressivo (es: Group02). Inoltre, per rendere più semplice la gestione delle animazioni e del cambio dei materiali si è ragionato sul porre, sempre riguardo la nomenclatura, due caratteri distintivi "*" per le animazioni e "-" per le mesh con materiali diversi. Questo è stato fatto per semplificare il processo di distinzione degli oggetti all’interno del codice.

Successivamente bisogna studiare quali sono i plugin necessari da aggiungere all’interno dell’applicazione. I plugins scelti sono i seguenti

```

1  await viewer.addPlugin(GBufferPlugin)
2  await viewer.addPlugin(new ProgressivePlugin(32))
3  const toneMapping = await viewer.addPlugin(new TonemapPlugin(false))
4  await viewer.addPlugin(GammaCorrectionPlugin)
5  await viewer.addPlugin(SSRPlugin)
6  await viewer.addPlugin(SSAOPlugin)
7  await viewer.addPlugin(BloomPlugin)
8  await viewer.addPlugin(RandomizedDirectionalLightPlugin, false)
9
10 const picking = await viewer.addPlugin(PickingPlugin);
11 const camViewPlugin = await viewer.addPlugin(CameraViewPlugin);
12 const gltfAnims = await viewer.addPlugin(GLTFAnimationPlugin);
```

Listing 3.1: File *index.ts* presente nel template di WebGi.

3.3 – SVILUPPO DEL PROGETTO

La loro scelta si è basata in base alla resa della qualità grafica e dei dettagli che si vogliono ottenere alla fine del rendering. Inoltre, i plugin relativi al picking, alla camera e alle animazioni, sono necessari per realizzare i requisti iniziali.

Tramite i materiali PBR ed i relativi plugin come clearcoat, parallax, anisotropy, l'HDR RGBM pipeline e i plugin di post processing, si è ottenuto un risultato molto simile ai render statici, andando a superare le aspettative iniziali.



Figura 3.1: Visualizzazione del mobile intero in WebGi.

L'immagine raffigura come l'applicazione risulta dopo aver aggiunto e impostato al meglio gli assets dei plugin relativi al render.

Attraverso un'osservazione più approfondita è possibile notare come alcuni degli effetti come Bloom, SSAO e SSR abbiano influito positivamente sul realismo della scena. Infatti, la rappresentazione di mobile per il makeup presenta materiali ed oggetti con caratteristiche particolarmente affini a questi effetti di post-processing.

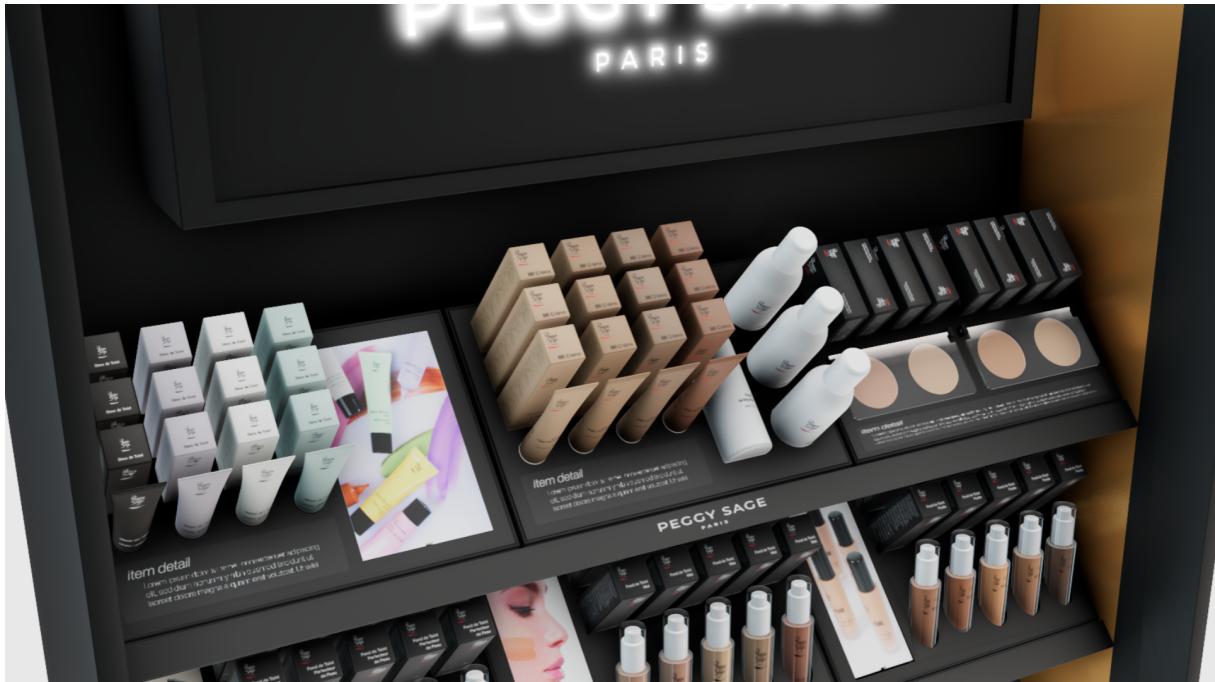


Figura 3.2: Resa grafica del SSAO e del Bloom sul mobile.

Il mobile presenta un led dietro ad un pannello, l'effetto ottenuto è di grande impatto e la fonte luminosa emessa è intensa e realistica.

Analogamente, l'ambient occlusion è sviluppata dinamicamente e si può notare al meglio fra le scatole centrali del primo ripiano. Se si guarda più attentamente l'immagine è possibile apprezzarla anche alla base dei prodotti e negli angoli fra il ripiano e il pannello laterale del mobile.



Figura 3.3: Visualizzazione dell'SSR sul mobile.

Oltre a poter apprezzare nuovamente l'SSAO, in questa immagine si può notare come gli specchi riflettano perfettamente le mesh circostanti, cambiando anche l'intensità di riflessione in base all'orientamento della camera. WebGi ha effettuato automaticamente il calcolo delle superfici riflettenti in base alle proprietà dei materiali. Solitamente, in altri framework, applicare le riflessioni ai materiali porta ad un rallentamento importante delle performance, con calo di fps e un incremento sull'utilizzo del processore. Grazie al progressive rendering e all'Adaptive Quality Control, la creazione delle riflessioni non aggrava le performance, infatti, la scena risulta essere fluida e di ottima qualità.

In questa immagine e nella precedente, è possibile apprezzare come i materiali PBR e quindi anche come il parallax mapping e la High Performance Refraction abbiano un impatto importante sul foto-realismo: si può notare come i vetrini emettano una lieve riflessione della luce e come generino una trasparenza leggermente opaca, rispecchiando un materiale reale come il petg, che è leggermente più opaco del vetro o del plexiglass.

Inoltre, se si analizza più attentamente il disco dietro al vetrino si può notare come il materiale non risulti piatto, ma come la sua superficie sembri presentare delle irregolarità dando un'impressione di ruvidità del materiale.

Passando alle funzionalità dell'applicazione, viene sviluppato per primo il picking degli



Figura 3.4: Materiali PBR con parallax mapping e reflections.

oggetti. Per il picking è stato utilizzato il PickingPlugin, attraverso la proprietà *hitObject* si riesce a capire quale mesh è stata toccata del click del mouse. Poiché in un oggetto sono presenti più mesh è stato necessario trovare il parent group della mesh, dal momento in cui il click del mouse colpisce una mesh dell'oggetto e non il gruppo intero.

```

1 picking!.autoFocus = true;
2
3 picking.addEventListener("hitObject", (e) => {
4   // change selection to parent
5   const selected = e.intersects.selectedObject.parent;
6
7   if (!focusMode) {
8     const parent = pickParent(selected);
9
10   if (isProduct(parent)) {
11     //save camera view
12     view = camViewPlugin.getCurrentCameraView(
13       viewer.scene.activeCamera
14     );
15     camViewPlugin.camViews[0] = view;
16
17   // change focus mode
18   focusMode = !focusMode;
19   picking!.autoFocus = true;
20
21   focus(parent);

```

3.3 – SVILUPPO DEL PROGETTO

```
22         //console.log(findProductMaterials(parent));
23     }
24 } else {
25     unFocus();
26 }
27
28 // remove skeleton grid of selection
29 e.intersects.selectedObject = null;
30});
```

Listing 3.2: Funzione per il picking degli oggetti.

La funzione pickParent() prende in input un oggetto 3D e risale, attraverso uno switch-case, fino al parent object. Una volta trovato il parent object si passa ad attivare la modalità focus, dove vengono rese invisibili tutte le mesh, eccetto quelle coinvolte dal picking.

Sul picking è stato deciso di rimuovere la colorazione dell'oggetto in base all'hovering, ovvero quando si va sopra con il mouse senza cliccare, funzionalità presente nel software sviluppato internamente all'azienda. Questa decisione è stata presa di comune accordo per testare se questa tipologia di interazione risultasse più intuitiva per l'utente. Infatti, gli basterebbe cliccare direttamente sul prodotto, eliminando quindi un passaggio prima di poterlo visualizzare nel dettaglio. Nel caso si volesse comunque aggiungere si può utilizzare il plugin di outline, che WebGi implementa come three.js, con la possibilità di colorare solo i contorni, oppure l'intera mesh.

```
1
2 async function focus(selectedObject: Object3D) {
3     isPlaying = false;
4     picking!.enabled = false;
5     viewer.scene.children[0].children[0].children.forEach((element) => {
6         if (element.name !== selectedObject.name) element.visible = false;
7     });
8
9     //disable pan
10    camera.controls!.enablePan = false;
11
12    await add_Infographics(selectedObject, hasAnimation(selectedObject));
13    findProductMaterials(selectedObject);
14    picking!.enabled = true;
15}
16
17 async function unFocus() {
```

```

19     focusMode = !focusMode;
20     gltfAnims!.resetAnimation();
21    .isPlaying = false;
22
23     viewer.scene.children[0].children[0].children.forEach((element) => {
24         element.visible = true;
25     });
26
27     await remove_Infographics();
28     restoreMaterial();
29
30     //enable pan
31     camera.controls!.maxDistance = 10;
32     camera.controls!.enablePan = true;
33
34     picking!.autoFocus = false;
35     await camViewPlugin.animateToView(camViewPlugin!.camViews[0], 2000);
36 }
```

Listing 3.3: Funzione per il focus e l'unfocus degli oggetti.

Durante la funzione focus viene posta visibile l'infografica attraverso la funzione *add-Infographics()* che prende come input l'oggetto e un boolean che utilizza per decidere se aggiungere i bottoni dell'animazione o meno. Analogamente, all'interno della funzione unfocus viene chiamata la funzione *remove-Infographics()* che rende invisibili i componenti HTML e ripristina la scena.

```

1 export async function add_Infographics(object: Object3D, hasAnimation: boolean) {
2     selectedObject = object;
3     switch (object.name) {
4         case "*Group01":
5             h_title.innerHTML = "PRODUCT";
6             prod_name.innerHTML = "OP GROUP\nProduct";
7             foto.style.backgroundImage = "";
8             p.innerText =
9                 "Product description";
10            shopRef = "";
11            break;
12        case "*Group02":
13            h_title.innerHTML = "PRODUCT";
14            prod_name.innerHTML = "OP GROUP\nProduct";
15            p.innerText =
16                "Product description";
17            foto.style.backgroundImage = "";
18            shopRef = "";
19            break;
```

3.3 – SVILUPPO DEL PROGETTO

```
20     }
21
22     materialsMenu?.classList?.add("show");
23     pause.classList.add("show");
24     stopAnim.classList.add("show");
25     shop.classList.add("show");
26     description.classList.add("show");
27     title.classList.add("show");
28     foto.classList.add("show");
29     playText.classList.add("show");
30     resetText.classList.add("show");
31     //check if the object has an animation and play it, also disable at the start the
32     //picking
33
34     shop.addEventListener("click", await shopping);
35
36     if (hasAnimation) {
37         pause.classList.add("show");
38         stopAnim.classList.add("show");
39     }
```

Listing 3.4: Funzione per rendere visibile l’infografica.

Dalla funzione si può notare il metodo usato per poter caricare dinamicamente l’infografica di un prodotto, questo metodo non è sicuramente il più efficiente ed ha senso di essere usato solo se vi sono pochi prodotti che hanno da mostrare le proprie caratteristiche.

Una soluzione per migliorare questo sistema è l’aggiunta di un database, sql o no sql, magari embeddato all’interno dell’applicazione o gestito su cloud, in modo da alleggerire il codice presente.

Infine, quando si esce dalla modalità di focus, la funzione unfocus() riporta la posizione e l’orientamento della camera ad una vista prefissata grazie al camViewPlugin.

Riguardo la gestione delle animazioni, se il prodotto ne presenta una, allora, compariranno i bottoni per la sua gestione. Si pongono gli elementi html dedicati in ascolto del click dell’utente e se ciò accade, attraverso il plugin GLTFAnimations, vengono chiamate le funzioni di *playPauseAnimations()* e *resetAnimation()* che rispettivamente mettono in pausa, riprendono e bloccano l’animazione.

```
1 const animation = document.getElementById("startAnimation") as HTMLElement;
2 const playText = document.getElementById("play-text") as HTMLElement;
3 animation.addEventListener("click", await pauseContinue);
4
```

```

5 const stop_animation = document.getElementById("stopAnimation") as HTMLElement;
6 stop_animation.addEventListener("click", await stopping);
7
8 ...
9
10 async function pauseContinue() {
11     gltfAnims!.loopRepetitions = 1;
12     gltfAnims!.playPauseAnimation();
13     if (!isPlaying) {
14         playText.innerHTML = "Pause";
15         isPlaying = true;
16     } else {
17         isPlaying = false;
18         playText.innerHTML = "Play";
19     }
20 }
21
22 async function stopping() {
23     if (isPlaying) {
24         await togglePlay();
25         isPlaying = false;
26     }
27     gltfAnims!.resetAnimation();
28 }
```

Listing 3.5: Gestione delle animazioni attraverso i bottoni.

Infine, quando viene attivata la modalità focus di un prodotto, si controlla se vi siano all'interno della scena, prodotti uguali, ma con un materiale diverso. Ad esempio, possono essere presenti dei fondotinta con tre gradazioni di colore. Per evitare di entrare ed uscire dalla modalità focus per visualizzare gli altri colori è stato aggiunta una interfaccia per poter cambiare colore direttamente dalla modalità focus.



Figura 3.5: Cambio del materiale attraverso i bottoni dell'infografica.

Viene sviluppato attraverso un controllo quando viene cliccato sul prodotto: se presenta una mesh che all'interno della scena è presente, con materiali diversi, viene fatto un traverse sulla scena e salvati tutti i materiali differenti applicati a quella mesh. Più nel dettaglio, all'interno della funzione viene fatto un traverse su tutta la scena fino a trovare le mesh corrispondenti, una volta trovate, viene salvato il materiale all'interno di un array, controllando che non sia già presente. Una volta ottenuto l'array con i materiali, viene aggiunto "show" al componente dell'infografica che toglierà il *display: none*; dallo stile e chiamerà la funzione *createMaterialUI* che effettua la conversione del colore del materiale in una stringa hex e andrà a creare un bottone con il colore corrispondente per ogni materiale presente nell'array creato in precedenza.

```

1 function createMaterialUI(materials : Material[] , mesh: Mesh){
2     oldMesh = mesh.clone();
3
4     materials.forEach((mat : any) => {
5         const li = document.createElement('li');
6         const span = document.createElement('span');
7         li.className = mat.name;
8         const colorSpan : Color = new Color;
9         if(mat.map === null) {
10             colorSpan.copy(mat.color);
11
12             span.style.background = '#'+colorSpan.convertLinearToSRGB().getHexString();

```

```

13     li.appendChild(span);
14
15     document?.querySelector('.materials--list')?.appendChild(li)
16     document?.querySelector("[id='"+li.className+"']")?
17       .addEventListener('click', () => {
18         changeMaterialColor(mat, mesh)
19         matIsChanged = true;
20         document?.querySelector('.materials--list'
21           .li.active)? .classList.remove('active')
22       })
23   );
24 }
25
26 function changeMaterialColor(mat: Material, mesh: Mesh){
27   mesh.material = mat;
28   newMesh = mesh;
29 }
30
31 export function restoreMaterial(){
32   if(matIsChanged){
33     newMesh.material = oldMesh.material;
34     matIsChanged = !matIsChanged
35   }
36 }
```

Listing 3.6: Funzione per la creazione e gestione del pannello di cambio dei materiali nell'UI.

Una volta creati gli elementi HTML, è stata aggiunta la gestione del click sul materiale: viene chiamata la funzione *changeMaterialColor()* che prende in ingresso un materiale ed una mesh e va semplicemente ad applicare quel materiale alla mesh.

Inoltre, è stata sviluppata la funzione *restoreMaterial()* che viene chiamata quando si esce dalla modalità focus e va a ripristinare i materiali originali della mesh coinvolta dal cambio.

Infine, è stata sviluppata la schermata di caricamento, il cui progresso avanza in base allo stato di avanzamento del caricamento della scena. Per farlo sono stati combinati la libreria di WebGi AssetImporter e la libreria di JavaScript gsap.

```

1 const importer = manager.importer as AssetImporter
2
3 importer.addEventListener("onProgress", (ev) => {
4   const progressRatio = (ev.loaded / ev.total)
```

3.3 – SVILUPPO DEL PROGETTO

```
5     document.querySelector('.progress')?.setAttribute('style', 'transform:
       scaleX(${progressRatio})')
6 })
7
8
9 importer.addEventListener("onLoad", (ev) => {
10     if(firstLoad){
11         introAnimation()
12     } else{
13         gsap.to('.loader', {x: '100%', duration: 0.8, ease: "power4.inOut", delay: 1})
14     }
15 })
16
17 function introAnimation(){
18     firstLoad = false
19     const introTL = gsap.timeline()
20     introTL.to('.loader', {opacity: 0, x: '100%', ease: "power4.inOut", delay: 0.4})
21     .fromTo('.webgi-canvas', {opacity: 0, y: '150%'}, {opacity: 1, y: '0%', ease:
22         "power4.inOut", duration: 1}, '-=1')
23     .fromTo('.webgi-canvas-container', {opacity: 0, x: '100%'}, {opacity: 1, x: '0%', ease: "power4.inOut", duration: 1.8}, '-=1')
24 }
```

Listing 3.7: Funzioni per la realizzazione della schermata di caricamento.

Attraverso la funzione `addEventListener()` dell'importer si riesce a distinguere l'inizio del caricamento dal progresso durante esso. Nel primo caso viene controllato che sia il primo caricamento della scena e viene chiamata la funzione `introAnimation()` che, sfruttando gsap, crea la progress bar. Durante il progresso, invece, attraverso css, si va ad aumentare la lunghezza della barra blu di caricamento, in relazione a quanto è stato caricato e quanto deve ancora essere fatto.

Questo è il risultato finale ottenuto.

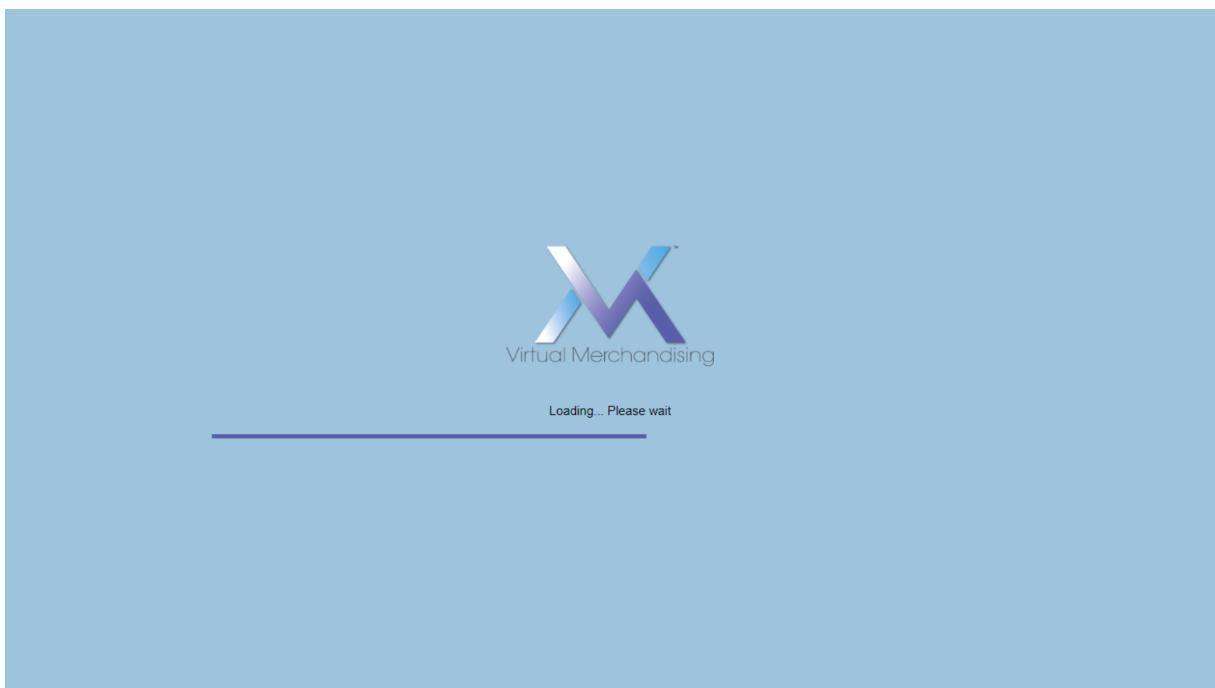


Figura 3.6: Schermata di caricamento della scena.

Per poi pubblicare in rete la scena viene fatto il build del codice sorgente con parcel e poi i file generati vengono caricati sul web tramite il provider dell’azienda.

Capitolo 4

Conclusioni

Al termine dello sviluppo dell'applicazione web e, dopo un confronto con l'azienda, è emerso che il framework risulta molto affidabile e in linea con le esigenze aziendali per la creazione di scene virtuali in realtime.

La pipeline di sviluppo risulta essere più snella e semplice da realizzare, ottimizzando il tempo di produzione del reparto grafico. L'infografica realizzata è ancora in una fase iniziale e può essere sicuramente migliorata ed adattata per la visualizzazione da mobile o tablet. Infine, la schermata di caricamento ha soddisfatto a pieno i requisiti iniziali.

In termini di performance e utilizzo di risorse, WebGi contiene internamente tutte le accortezze necessarie per ottimizzare le performance. Rispetto al software precedente ci sono alcune differenze sostanziali: il tempo di caricamento drasticamente ridotto, una resa grafica più realistica e una fluidità di interazione nettamente migliore, anche con macchine meno performanti in termini di CPU e GPU.

Analizzando più nel dettaglio le differenze di resa grafica si può notare che, con il vecchio software alcuni materiali come il vetro, gli specchi o i metalli non erano resi al meglio. Ora, attraverso i materiali PBR e i plugin di post-processing si è giunti al realismo desiderato.

Riguardo le performance la vecchia scena viaggiava a massimo 30fps, senza post-processing e a 11/12fps con il post-processing attivo. Con l'uso di WebGi si raggiungono in media 59/60fps, nonostante siano attivi tutti gli algoritmi di post-processing. Questo grazie al progressive rendering e le ottimizzazioni effettuate in base al dispositivo.

4.1 Limiti

In termini qualitativi WebGi risulta essere la soluzione ideale per un’azienda che crea scene virtuali in realtime e, considerando anche le feature in fase di sviluppo, come l’Augmented Reality (AR), può dare molti input per realizzare progetti eccellenti e con facilità.

Il limite principale però, è il prezzo, che, per poter sfruttare WebGi per fini commerciali, risulta completamente disallineato dal mercato. Rispetto ai costi proposti dai competitor, il costo risulta essere dieci volte superiore. Va inoltre tenuto in considerazione che WebGi è attualmente in Beta e quindi non è un prodotto ancora affermato sul mercato.

Un altro problema riscontrato, è la difficoltà della creazione di plugin personalizzati, non è processo immediato per chi non ha esperienza in TypeScript, sviluppo di scene virtuali in three.js e shader in WebGL.

4.2 Sviluppi futuri

L’applicazione web sviluppata risulta essere in una fase iniziale e vi sono migliorie che possono essere applicate.

Inizialmente, la gestione del caricamento del contenuto dell’infografica può essere gestito attraverso un database, si potrebbe utilizzare firebase, database noSQL sviluppato da Google che, attraverso documenti e collezioni di documenti, un’interfaccia semplice, può essere popolato direttamente da un’impiegato dell’azienda, senza particolari difficoltà. Fornisce un piano gratuito e una tariffa molto economica, una volta che vengono sforate le soglie di scrittura e lettura sul DB.

Inoltre, è possibile rendere responsive l’interfaccia grafica, rendendo l’infografica e la visualizzazione della scena adatte ad un telefono e un tablet. Attraverso CSS è possibile selezionare la larghezza e l’altezza della finestra di visualizzazione del dispositivo e di conseguenza modificare lo stile e la posizione dei componenti. Infine, per poter migliorare la resa grafica, sarebbe possibile scrivere dei plugin personalizzati per poter rappresentare elementi molto complicati da rappresentare come i fluidi. Infatti, in ambito makeup è molto frequente il voler rappresentare un liquido all’interno di un prodotto, oppure realizzare animazioni che lo comprendono. Si potrebbe sviluppare uno shader dedicato e poi creare il relativo plugin.

Bibliografia

- [1] Autodesk. URL: <https://knowledge.autodesk.com/it/search-result/caas/CloudHelp/clouddhelp/ITA/ARENDERING/files/GUID-50A91891-EC13-4A50-9D9F-95E532904FC7-htm.html>.
- [2] Autodesk. URL: <https://knowledge.autodesk.com/it/support/autocad/learn-explore/caas/CloudHelp/clouddhelp/2019/ITA/AutoCAD-Core/files/GUID-A6232957-5039-4AB7-8B1D-8FD0AD98F77B-htm.html>.
- [3] Babylonjs. URL: <https://doc.babylonjs.com/features/featuresDeepDive/materials/using/parallaxMapping>.
- [4] Blender. URL: <https://docs.blender.org/manual/en/latest/modeling/meshes/editing/uv.html>.
- [5] CommunityArm. URL: <https://community.arm.com/arm-community-blogs/b/graphics-gaming-and-vr-blog/posts/high-quality-rgbm-texture-compression-with-astc>.
- [6] Everyeye. URL: <https://www.everyeye.it/articoli/speciale-bump-normal-mapping-5989.html>.
- [7] Everyeye. URL: <https://tech.everyeye.it/articoli/speciale-antialiasing-tipologie-differenze-prestazioni-delle-tecniche-aa-31738.html>.
- [8] Everyeye. URL: <https://www.everyeye.it/articoli/speciale-cg-effect-il-bloom-5969.html>.
- [9] Google. URL: <https://codelabs.developers.google.com/codelabs/draco-3d#0>.
- [10] Hdrsoft. URL: <https://www.hdrsoft.com/resources/dri.html#hdri>.

- [11] Brian Karis. URL: https://de45xmedrsdbp.cloudfront.net/Resources/files/TemporalAA_small-59732822.pdf.
- [12] Learopengl. URL: <https://learnopengl.com/PBR/Theory>.
- [13] Learopengl. URL: <https://learnopengl.com/Advanced-Lighting/SSAO>.
- [14] Lettier.Github. URL: <https://lettier.github.io/3d-game-shaders-for-beginners/screen-space-reflection.html>.
- [15] Microservices. URL: <https://microservices.io/patterns/apigateway.html>.
- [16] Pcmag. URL: <https://www.pc当地.com/encyclopedia/term/jitter>.
- [17] Polycount. URL: http://wiki.polycount.com/wiki/Anisotropic_map.
- [18] SmartWorld. URL: <https://www.smartworld.it/informatica/ray-tracing-spiegato.html#cose>.
- [19] Three.js. URL: <https://threejs.org/docs/#api/en/core/Raycaster>.
- [20] Three.js. URL: <https://threejs.org/docs/#api/it/objects/LOD>.
- [21] *Toward Evaluating Progressive Rendering Methods in Appearance Design Tasks*. 2012. URL: <https://cgg.mff.cuni.cz/~jaroslav/papers/2012-renderstudy/>.
- [22] Tutorialspoint. URL: https://www.tutorialspoint.com/webgl/webgl_graphics_pipeline.htm.
- [23] Unity3d. URL: <https://docs.unity3d.com/560/Documentation/Manual/PostProcessingOverview.html>.
- [24] WebGi. URL: <https://webgi.xyz/docs/manual/viewer-api>.
- [25] WebGi. URL: <https://webgi.xyz>.
- [26] WebGi. URL: <https://webgi.xyz/docs/manual/custom-plugins>.
- [27] Webgi. URL: <https://webgi.xyz/docs/manual/asset-management>.
- [28] Wikipedia. URL: https://en.wikipedia.org/wiki/Glossary_of_computer_graphics#screen_space.
- [29] Wikipedia. URL: https://it.wikipedia.org/wiki/Parallax_mapping.
- [30] Wikipedia. URL: https://en.wikipedia.org/wiki/Tone_mapping.

BIBLIOGRAFIA

- [31] Wikipedia. URL: https://en.wikipedia.org/wiki/Screen_space_ambient_occlusion.
- [32] Wikipedia. URL: [https://it.wikipedia.org/wiki/Intercalazione_\(animazione\)](https://it.wikipedia.org/wiki/Intercalazione_(animazione)).
- [33] Ziyadbarakat. URL: <https://ziyadbarakat.wordpress.com/2020/07/28/temporal-anti-aliasing-step-by-step/>.