ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Dipartimento di Informatica - Scienza e Ingegneria

Corso di Laurea in Ingegneria e Scienze Informatiche

Illuminazione e Shading in Tempo Reale con Unity: dai Modelli Classici al Physically Based Rendering

Relatore:
Prof.ssa Damiana Lazzaro

Presentata da:
Ciro Bassi

Sessione Unica
Anno Accademico 2025/2026

Indice

Elenco delle figure

Elenco delle tabelle

Elenco dei codici

Introduzione

La sintesi di immagini tridimensionali fotorealistiche è da sempre uno degli obiettivi principali della computer graphics. Alla base di qualsiasi immagine generata in tempo reale c'è la *pipeline di rendering*: un processo complesso che coinvolge la geometria della scena, le proprietà dei materiali e il comportamento della luce.

Negli ultimi decenni, l'evoluzione dell'hardware grafico ha reso possibile eseguire questo processo a frequenze interattive, migliorando l'esperienza in applicazioni videoludiche e non solo. Parallelamente, i modelli matematici per descrivere l'interazione tra luce e materia sono stati raffinati: dai modelli empirici di Phong e Blinn-Phong fino ai moderni modelli di *Physically Based Rendering* (PBR), che approssimano più fedelmente il comportamento della luce.

La scelta del modello di illuminazione ha un impatto diretto sulla qualità visiva e sulle prestazioni di un'applicazione in tempo reale. Un modello empirico come Blinn-Phong è computazionalmente economico e intuitivo, ma produce risultati che si discostano sensibilmente dalla realtà. Il PBR, al contrario, garantisce una risposta visiva coerente in qualsiasi contesto, a fronte di un costo computazionale più elevato. Comprendere le differenze tra questi approcci è fondamentale per operare nel campo della grafica interattiva.

L'obiettivo di questo elaborato è comparare i principali modelli di illuminazione utilizzati nella grafica in tempo reale, implementandoli in HLSL all'interno del motore Unity. A tal fine è stata adottata la *Universal Render Pipeline* (URP), una pipeline leggera e versatile che consente di scrivere shader personalizzati con pieno controllo sul calcolo dell'illuminazione, rendendola ideale per implementare e confrontare

direttamente i modelli trattati. Il lavoro copre la pipeline di rendering programmabile, i modelli classici di Phong e Blinn-Phong, il Physically Based Rendering con la BRDF di Cook-Torrance, alcuni shader personalizzati a scopo dimostrativo e un progetto in Unity che mette a confronto visivamente i modelli implementati.

La tesi è così strutturata:

- il capitolo 1 introduce la pipeline di rendering programmabile nei suoi stadi principali e fornisce le basi di Unity e del linguaggio HLSL;
- il capitolo 2 tratta la teoria dei modelli di illuminazione classici, Phong e Blinn-Phong, e delle tecniche di shading (flat, Gouraud e Phong shading);
- il capitolo 3 è dedicato alla teoria del Physically Based Rendering: i principi fisici alla base del modello, la BRDF di Cook-Torrance, il workflow metallic/-roughness e un'introduzione all'Image-Based Lighting;
- il capitolo 4 presenta, dal punto di vista concettuale, alcuni shader personalizzati che esulano dal fotorealismo, mostrando le possibilità espressive della pipeline programmabile;
- il capitolo 5 raccoglie l'implementazione HLSL di tutti gli shader, la loro integrazione nel progetto Unity, le scene di test e il confronto visivo e prestazionale tra i modelli;
- il capitolo 6 presenta le considerazioni conclusive, sintetizzando i risultati ottenuti, i limiti dell'implementazione e le possibili direzioni di sviluppo futuro.

Capitolo 1

La pipeline di rendering

La computer graphics ha come obiettivo principale la generazione di immagini a partire da una descrizione matematica di una scena tridimensionale. Tale descrizione, costituita da geometrie, materiali, luci e parametri della camera, viene trasformata progressivamente in un'immagine bidimensionale destinata alla visualizzazione su schermo. L'insieme delle fasi che realizzano questa trasformazione prende il nome di *pipeline di rendering*. Nelle architetture moderne, gran parte di questo processo è implementato direttamente in hardware dalle GPU (*Graphics Processing Unit*), che eseguono in modo altamente parallelo le operazioni necessarie alla produzione dei pixel finali.

Questo capitolo introduce la struttura generale della pipeline grafica programmabile, descrivendo i suoi stadi principali e il loro ruolo nel processo di rendering.

1.1 La pipeline di rendering programmabile

Le prime GPU erano basate su una pipeline non programmabile (*fixed-function pipeline*): ogni stadio del processo di rendering eseguiva operazioni rigidamente predefinite a livello hardware, senza alcuna possibilità di personalizzazione da parte dello sviluppatore. Questo approccio rendeva l'implementazione di effetti grafici non standard estremamente complessa, se non del tutto impossibile.

A partire dai primi anni 2000, con l'introduzione delle prime GPU programmabili, alcune fasi della pipeline divennero configurabili e personalizzabili dallo sviluppatore. Le GPU iniziarono a integrare unità computazionali dedicate, dette *Shader Units*, riprogrammabili tramite appositi programmi chiamati *shaders*. Gli shader, scritti direttamente dallo sviluppatore, vengono eseguiti in parallelo sulla GPU per ogni vertice o frammento della scena. Questo approccio ha reso possibile implementare algoritmi grafici personalizzati, superando i limiti imposti dalla pipeline non programmabile.

La pipeline grafica moderna, ottimizzata nella quasi totalità dei casi per l'elaborazione di triangoli, è composta da una sequenza di stadi che possono essere raggruppati in tre fasi concettuali:

1. **Fase applicativa**: eseguita sulla CPU, si occupa di preparare i dati della scena (geometrie, parametri dei materiali, ...) e di inviarli alla GPU.

2. **Fase geometrica**: opera sui vertici della scena, trasformandone le coordinate dallo spazio locale dell'oggetto a quello del mondo, fino allo spazio di proiezione della camera. Comprende il *vertex shader* e, opzionalmente, altri stadi tra cui *tessellation* e *geometry shader*.

3. **Fase di rasterizzazione e shading**: converte le primitive geometriche in frammenti (potenziali pixel), ne calcola il colore finale attraverso il *fragment shader*, e determina quali frammenti vengono effettivamente visualizzati a schermo tramite l'*output merger*.

1.2 Stadi principali della pipeline

Di seguito vengono descritti i principali stadi della pipeline. Tra questi, il vertex shader e il fragment shader rappresentano le principali fasi programmabili: è infatti al loro interno che è possibile definire il comportamento della geometria e calcolare il colore finale dei frammenti.

1.2.1 Vertex shader

Il *vertex shader* è il primo stadio programmabile della pipeline. Viene eseguito una volta per ogni vertice della geometria inviata alla GPU e ha come compito principale quello di trasformarne le coordinate attraverso una serie di spazi di riferimento, fino a ottenere la posizione finale utilizzata nelle fasi successive della pipeline:

1. **Object space**: rappresenta lo spazio locale del modello 3D. In questo sistema di riferimento, le coordinate del vertice sono definite rispetto all'origine e all'orientamento dell'oggetto stesso.

2. **World space**: rappresenta lo spazio globale della scena. Le coordinate del vertice vengono trasformate in base alla posizione, alla rotazione e alla scala dell'oggetto, mediante la matrice di modello $\mathbf{M}$.

3. **View space**: rappresenta lo spazio della camera. Le coordinate sono espresse rispetto al punto di vista dell'osservatore, mediante la matrice di vista $\mathbf{V}$.

4. **Clip space**: rappresenta lo spazio ottenuto dopo l'applicazione della proiezione. Le coordinate vengono trasformate mediante la matrice di proiezione $\mathbf{P}$, preparando la geometria alle successive fasi di clipping, divisione prospettica e rasterizzazione.

La trasformazione complessiva di un vertice $\mathbf{v}$ dalla posizione in object space alla posizione in clip space è data dal prodotto:

$$\mathbf{v}_{\text{clip}} = \mathbf{P} \cdot \mathbf{V} \cdot \mathbf{M} \cdot \mathbf{v} \tag{1.1}$$

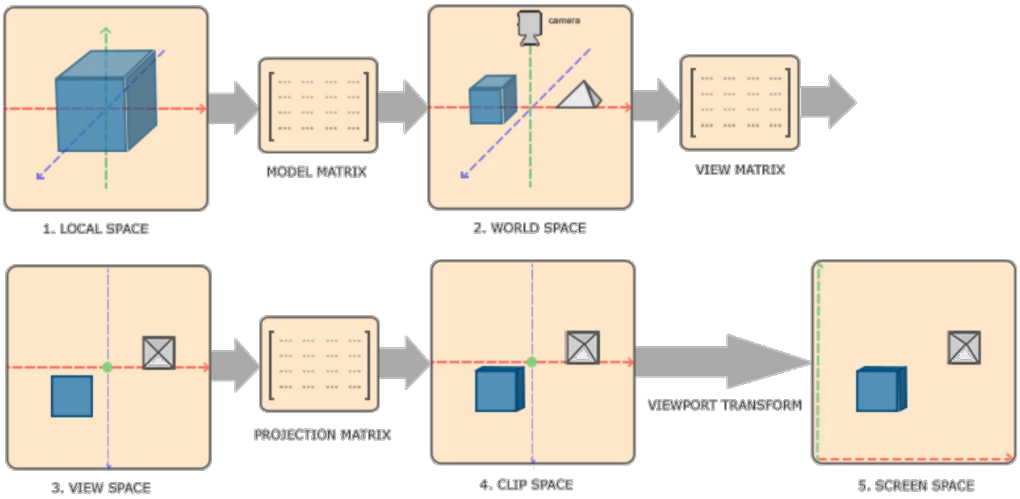


Figura 1.1: La sequenza di trasformazioni delle coordinate: dallo spazio locale dell'oggetto (*object space*), allo spazio del mondo (*world space*), allo spazio della camera (*view space*), fino allo spazio di proiezione (*clip space*).

Oltre alla posizione, il vertex shader può ricevere in ingresso e trasformare altri attributi associati al vertice, come la normale **N**, fondamentale per i calcoli di illuminazione, o le coordinate texture, necessarie per il campionamento delle immagini applicate alla superficie.

I dati prodotti dal vertex shader, detti *varyings*, vengono successivamente interpolati durante la fase di rasterizzazione prima di essere forniti in ingresso al fragment shader.

1.2.2 Rasterizzazione

La rasterizzazione è lo stadio, non programmabile, che converte le primitive geometriche (triangoli definiti dai loro vertici in clip space, ottenuti dal vertex shader) in un insieme di frammenti, corrispondenti ciascuno a un potenziale pixel sullo schermo.

Per ogni frammento generato, la rasterizzazione interpola linearmente gli attributi calcolati dal vertex shader nei vertici del triangolo corrispondente.

1.2.3 Fragment shader

Il *fragment shader* è il secondo dei principali stadi programmabili della pipeline. Esso viene eseguito una volta per ciascun frammento generato dalla rasterizzazione e ha il compito di determinarne il colore finale. Per svolgere questa operazione, il fragment shader riceve in ingresso i dati interpolati durante la fase di rasterizzazione, come la normale, la posizione nello spazio e le coordinate texture. Tali informazioni vengono utilizzate per valutare il modello di illuminazione scelto e calcolare il contributo della luce sulla superficie. I modelli di illuminazione vengono solitamente implementati in questo stadio, anziché nel vertex shader, poiché il calcolo dell'illuminazione per-pixel (*Phong shading*) produce risultati significativamente più accurati rispetto al calcolo per-vertice, soprattutto in presenza di riflessi speculari o variazioni locali della superficie.

1.2.4 Output merger

L'*output merger* è l'ultimo stadio della pipeline, non programmabile. Riceve i frammenti prodotti dal fragment shader e decide quali vengono effettivamente scritti nel framebuffer (il buffer che contiene l'immagine finale destinata allo schermo). Questa decisione avviene attraverso una serie di operazioni, tra cui il *depth test*, che scarta i frammenti nascosti da geometrie più vicine alla camera, e il *blending*, che combina il colore del frammento con quello già presente nel framebuffer. Quest'ultima operazione è particolarmente importante nella gestione della trasparenza.

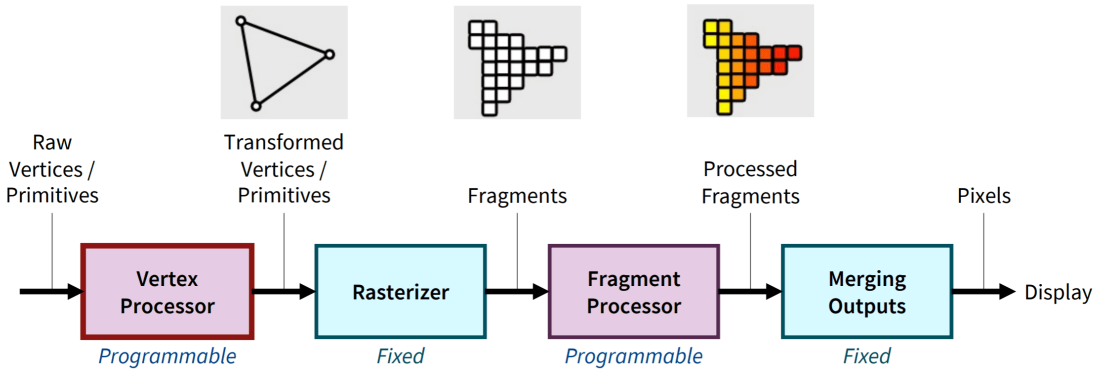


Figura 1.2: I principali stadi della pipeline di rendering.

1.3 La pipeline di rendering in Unity

Unity è un motore di gioco che supporta diverse pipeline di rendering attraverso il sistema *Scriptable Render Pipeline*. Questo consente di definire l'intero flusso di rendering, offrendo un livello di controllo che la precedente pipeline integrata (*Built-in Render Pipeline*) non permetteva.

Le due pipeline attualmente fornite da Unity sono la *Universal Render Pipeline* (URP), orientata alle prestazioni e alla portabilità, e la *High Definition Render Pipeline* (HDRP), orientata alla massima qualità visiva su hardware di fascia alta.

Nel progetto sviluppato in questa tesi è stata adottata URP: poiché l'obiettivo principale consiste nell'implementazione manuale dei modelli di illuminazione, era necessario utilizzare una pipeline che offrisse un controllo diretto sul codice di rendering, riducendo al minimo la complessità aggiuntiva. In questo modo è stato possibile concentrare l'attenzione sulla logica di illuminazione e sul confronto tra i modelli implementati. La scelta della pipeline non influenza i fondamenti teorici trattati: i modelli, le tecniche di shading e la struttura della pipeline grafica rimangono concettualmente identici indipendentemente dalla pipeline adottata.

1.4 Il linguaggio HLSL e ShaderLab

Gli shader in Unity vengono scritti in file con estensione `.shader` che utilizzano ShaderLab, un linguaggio proprietario che descrive la struttura generale dello shader e le sue proprietà. All'interno di questa struttura viene inserito il codice HLSL, responsabile dell'implementazione della logica del vertex shader e del fragment shader.

Uno shader Unity è organizzato nei seguenti blocchi: `Properties`, che espone i parametri del materiale, configurabili nell'inspector di Unity; `Shader`, il blocco esterno identificato da un nome; uno o più `SubShader`, ciascuno pensato per una specifica piattaforma; uno o più `Pass`, che rappresentano singole esecuzioni della pipeline, ognuna con il proprio vertex e fragment shader.

L'esempio seguente mostra la struttura minima di uno shader per URP, con un vertex shader che trasforma le coordinate e un fragment shader che restituisce un colore uniforme:

Listing 1.1: Struttura minima di uno shader personalizzato per URP.

```
Shader "Custom/NomeShader" {
   Properties { // Proprieta' configurabili dall'inspector di Unity
      _Color ("Color", Color) = (1, 1, 1, 1)
   }

   SubShader {
      Tags { "RenderPipeline"="UniversalPipeline" }

      Pass {
         Tags { "LightMode"="UniversalForward" }

         HLSLPROGRAM // Inizio del codice HLSL

         #pragma vertex vert
         #pragma fragment frag

         #include
             "Packages/com.unity.render-pipelines.universal/ShaderLibrary/Core.hlsl"

         struct Attributes { // Dati in input al vertex shader
            float4 positionOS : POSITION;
         };

         struct Varyings { // Dati in input al fragment shader (interpolati)
            float4 positionCS : SV_POSITION;
         };

         float4 _Color; // Rende la property accessibile al codice HLSL

         Varyings vert(Attributes IN) { // Vertex shader
            Varyings OUT;
            OUT.positionCS = TransformObjectToHClip(IN.positionOS.xyz);
            return OUT;
         }

         float4 frag(Varyings IN) : SV_Target { // Fragment shader
            return _Color;
         }

         ENDHLSL // Fine del codice HLSL
      }
   }
}
```

Le strutture `Attributes` e `Varyings` definiscono rispettivamente i dati in ingresso al vertex shader e quelli passati al fragment shader dopo la rasterizzazione. Ogni campo è annotato con una semantica (`POSITION`, `NORMAL`, ...) che indica alla GPU il significato del dato e come instradarlo nella pipeline. I suffissi `OS`, `WS` e `CS` sono una convenzione di Unity per indicare lo spazio di riferimento del valore: *Object Space*, *World Space* e *Clip Space*.

Questa struttura è la base su cui verranno costruiti tutti gli shader nei capitoli successivi.

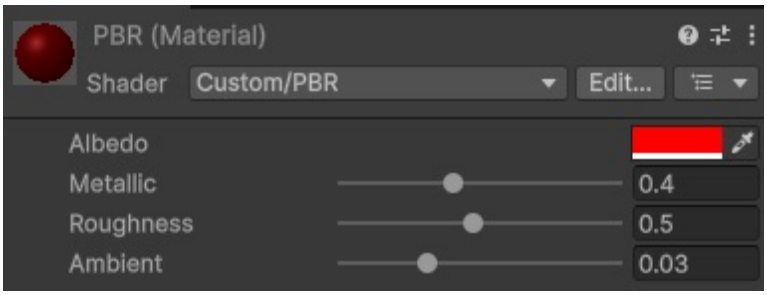


Figura 1.3: L'inspector di un materiale in Unity: i parametri mostrati corrispondono alle proprietà dichiarate nel blocco `Properties` dello shader.

Capitolo 2

Modelli classici di illuminazione e tecniche di shading

Questo capitolo presenta i modelli di illuminazione storicamente più rilevanti nella grafica in tempo reale: il modello di Phong e la sua variante proposta da Blinn. Si tratta di modelli empirici: pur non essendo fondati su una descrizione fisicamente accurata dell'interazione tra luce e materia, approssimano in modo efficace il comportamento visivo della luce, mantenendo un costo computazionale contenuto.

Entrambi sono modelli di illuminazione locale: calcolano il colore di un punto considerando solo la luce che arriva direttamente dalle sorgenti, senza tener conto dei rimbalzi successivi tra le superfici. Al contrario, i modelli di illuminazione globale simulano anche la luce indiretta producendo risultati molto più realistici, ma a un costo computazionale spesso incompatibile con il rendering in tempo reale e non verranno trattati in questa tesi. Nel capitolo 3 verrà discusso come, nell'ambito del rendering fisicamente basato, la luce indiretta possa essere approssimata mediante tecniche come l'*Image-Based Lighting*.

Dopo aver introdotto i concetti di base sull'interazione tra luce e superficie, vengono formalizzati i due modelli e analizzate le differenze tra le principali tecniche di shading: flat, Gouraud e Phong shading. Le relative implementazioni HLSL sono presentate nel capitolo 5, dedicato alla parte pratica del lavoro.

2.1 Interazione tra luce e superficie

Per comprendere i modelli di illuminazione è necessario introdurne i concetti fisici alla base dell'interazione tra luce e superficie, analizzando sia il comportamento della luce quando incide su un materiale, sia il modo in cui la risposta della superficie può essere scomposta in componenti più semplici da modellare.

2.1.1 Componenti della luce riflessa

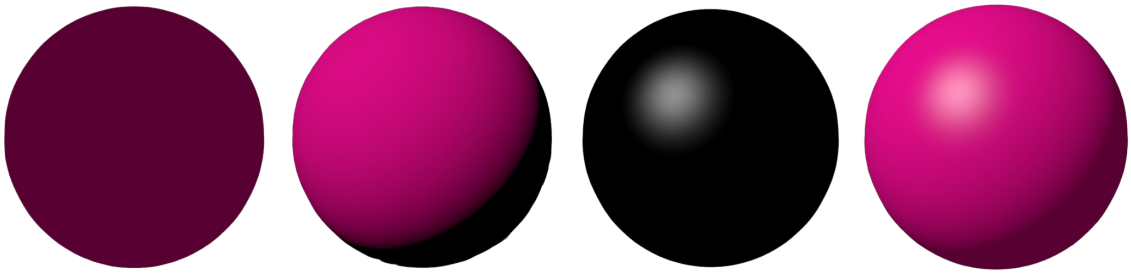
Quando un raggio di luce colpisce una superficie opaca, la sua energia viene in parte assorbita e in parte riflessa. La componente riflessa può essere descritta come la somma di tre contributi distinti:

- **Componente ambientale**: rappresenta la luce indiretta, ovvero quella che ha rimbalzato più volte nell'ambiente circostante prima di raggiungere la superficie. Nei modelli classici viene approssimata con un termine costante: una stima grossolana, ma evita che le zone in ombra appaiano completamente nere.
- **Componente diffusa**: rappresenta la luce riemessa dalla superficie uniformemente in tutte le direzioni. La sua intensità non dipende dalla posizione dell'osservatore, bensì dall'angolo tra la direzione della luce incidente e la normale alla superficie.
- **Componente speculare**: rappresenta la luce riflessa direzionalmente dalla superficie. A differenza della componente diffusa, la sua intensità dipende anche dalla posizione dell'osservatore: il riflesso speculare è visibile solo quando l'occhio si trova in prossimità della direzione di riflessione. La forma e l'intensità di questo riflesso dipendono dalle caratteristiche del materiale: un metallo lucido produce un riflesso piccolo e intenso, mentre una plastica opaca ne produce uno ampio e debole.

Il colore finale di un punto sulla superficie viene calcolato come la somma delle tre componenti:

$$I = I_a + I_d + I_s \tag{2.1}$$

dove I_a, I_d e I_s rappresentano rispettivamente le componenti ambientale, diffusa e speculare.



(a) Ambientale **(b)** Diffusa **(c)** Speculare **(d)** Risultato finale

Figura 2.1: Le tre componenti della luce riflessa su una sfera e la loro somma.

2.1.2 Sorgenti luminose

Esistono diverse tipologie di sorgenti luminose (direzionali, puntuali e spot) che si differenziano per geometria e comportamento. Negli shader sviluppati successivamente verrà utilizzata, per semplicità, esclusivamente una luce direzionale, che modella una sorgente infinitamente lontana con raggi luminosi paralleli, una tipica approssimazione della luce solare.

2.2 Il modello di Phong

Il modello di Phong, proposto da Bui Tuong Phong nel 1975 [1], è un modello di illuminazione locale ed è uno dei primi modelli empirici per la sintesi di immagini tridimensionali. Pur non essendo fisicamente accurato, produce risultati visivamente convincenti a costo computazionale contenuto, ed è rimasto a lungo il riferimento principale nella grafica in tempo reale.

Il modello si basa sulla decomposizione della luce riflessa nelle tre componenti introdotte nella sezione 2.1.1: ambientale, diffusa e speculare.

Per comprenderne la formulazione matematica è necessario definire i principali vettori unitari coinvolti nel calcolo dell'illuminazione, valutati nel punto della superficie considerato:

- **N**: normale alla superficie, diretta verso l'esterno;
- **L**: direzione verso la sorgente luminosa;
- **V**: direzione verso l'osservatore;
- **R**: direzione di riflessione di **L** rispetto a **N**, calcolata come:

$$\mathbf{R} = 2(\mathbf{N} \cdot \mathbf{L})\,\mathbf{N} - \mathbf{L} \tag{2.2}$$

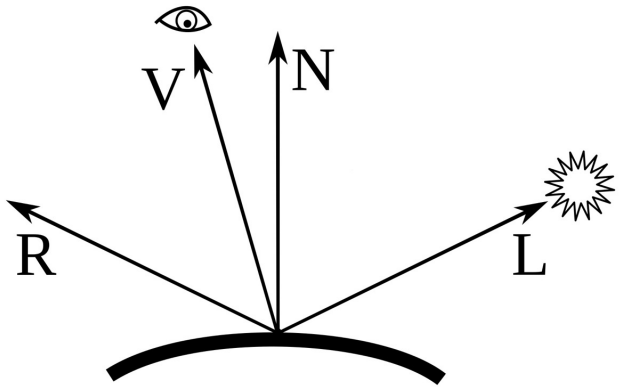


Figura 2.2: Vettori coinvolti nel modello di Phong.

2.2.1 Componente ambientale

L'intensità della componente ambientale è data da:

$$I_a = k_a \cdot c \tag{2.3}$$

dove $k_a \in [0, 1]$ è il coefficiente di riflessione ambientale e c è il colore del materiale.

2.2.2 Componente diffusa

L'intensità della componente diffusa segue la legge di Lambert: dipende dal coseno dell'angolo tra **N** e **L**. Poiché entrambi i vettori sono unitari, il coseno corrisponde

al loro prodotto scalare:

$$I_d = k_d \cdot (\mathbf{N} \cdot \mathbf{L}) \cdot c \cdot c_L \tag{2.4}$$

dove $k_d \in [0, 1]$ è il coefficiente di riflessione diffusa, c è il colore del materiale e c_L è il colore della sorgente luminosa. Nell'implementazione il prodotto scalare viene limitato a zero per evitare valori negativi quando la superficie è rivolta in direzione opposta alla luce.

2.2.3 Componente speculare

L'intensità della componente speculare dipende dall'angolo tra la direzione di riflessione $\mathbf{R}$ e la direzione di vista $\mathbf{V}$:

$$I_s = k_s \cdot (\mathbf{R} \cdot \mathbf{V})^n \cdot c_L \tag{2.5}$$

dove $k_s \in [0, 1]$ è il coefficiente di riflessione speculare, n è l'esponente di lucentezza (*shininess*) e c_L è il colore della sorgente luminosa. La componente diffusa e quella speculare dipendono entrambe dal colore della luce, ma in modo diverso: la diffusa è modulata anche dal colore del materiale, poiché rappresenta la luce che penetra nella superficie e viene riemessa, mentre la speculare assume il solo colore della luce, trattandosi di luce riflessa direttamente senza essere assorbita. La shininess n controlla l'ampiezza del riflesso: valori bassi producono riflessi ampi e morbidi, mentre valori alti producono riflessi piccoli e concentrati.

2.2.4 Formula completa

La formula finale del modello di Phong è data dalla somma delle tre componenti:

$$I = k_a \cdot c + k_d \cdot (\mathbf{N} \cdot \mathbf{L}) \cdot c \cdot c_L + k_s \cdot (\mathbf{R} \cdot \mathbf{V})^n \cdot c_L \tag{2.6}$$

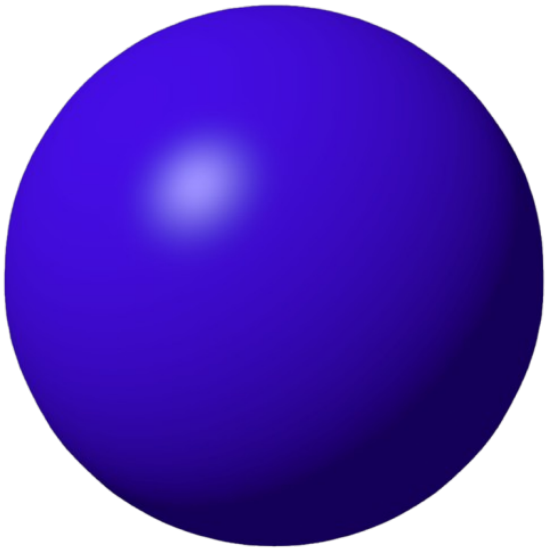


Figura 2.3: Una sfera renderizzata con il modello di Phong.

2.2.5 Limiti del modello

Nonostante la sua semplicità ed efficacia, il modello di Phong presenta alcuni limiti significativi:

- **Assenza di conservazione dell'energia**: nessun vincolo impedisce che la somma dei coefficienti $k_a + k_d + k_s$ superi 1, producendo più luce riflessa di quella incidente.
- **Il modello è empirico**: le tre componenti non derivano da principi fisici ma da un'osservazione qualitativa del comportamento della luce. Il risultato è visivamente plausibile ma non fisicamente corretto.
- **Riflesso speculare troncato**: quando l'angolo tra **R** e **V** supera i 90°, il prodotto scalare diventa negativo e viene annullato (poiché limitato a zero). In alcuni casi questo produce un taglio netto del riflesso speculare (figura 5.6).

L'ultimo di questi limiti è uno dei motivi principali dell'introduzione del modello di Blinn-Phong.

2.3 Il modello di Blinn-Phong

Nel 1977 James Blinn [2] propose una variante del modello di Phong che ne modifica la componente speculare, mantenendo invariate le componenti ambientale e diffusa.

2.3.1 L'halfway vector

La modifica introdotta da Blinn consiste nel sostituire il vettore di riflessione **R** con l'*halfway vector* **H**, definito come il vettore a metà tra la direzione della luce **L** e la direzione di vista **V**:

$$\mathbf{H} = \frac{\mathbf{L} + \mathbf{V}}{\|\mathbf{L} + \mathbf{V}\|} \tag{2.7}$$

Si osserva che una superficie riflette la luce in modo speculare verso l'osservatore quando la sua normale è allineata con **H**. La componente speculare viene quindi riscritta come:

$$I_s = k_s \cdot (\mathbf{N} \cdot \mathbf{H})^n \cdot c_L \tag{2.8}$$

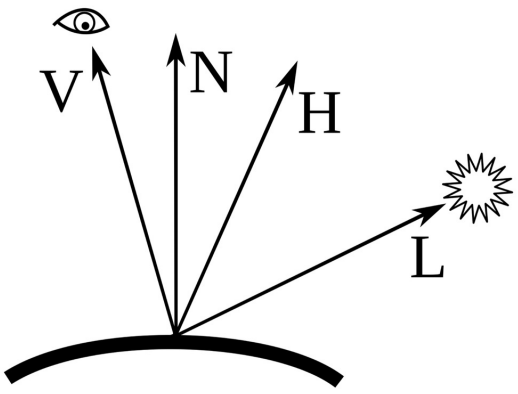


Figura 2.4: Vettori coinvolti nel modello di Blinn-Phong.

Poiché l'angolo tra $\mathbf{N}$ e $\mathbf{H}$ è circa la metà dell'angolo tra $\mathbf{R}$ e $\mathbf{V}$, a parità di esponente n il modello di Blinn-Phong produce un riflesso speculare più ampio rispetto a Phong. Per ottenere un riflesso di dimensione simile è necessario utilizzare un esponente circa quattro volte maggiore.

2.3.2 Formula completa

Le componenti ambientale e diffusa rimangono invariate rispetto al modello di Phong. La formula completa di Blinn-Phong è quindi:

$$I = k_a \cdot c + k_d \cdot (\mathbf{N} \cdot \mathbf{L}) \cdot c \cdot c_L + k_s \cdot (\mathbf{N} \cdot \mathbf{H})^n \cdot c_L \tag{2.9}$$

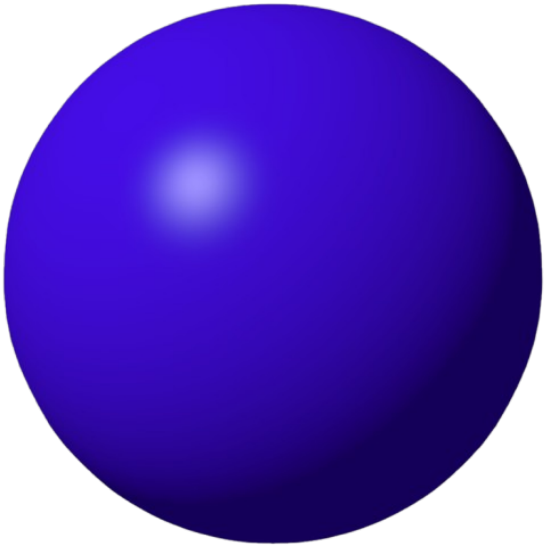


Figura 2.5: Una sfera renderizzata con il modello di Blinn-Phong.

2.3.3 Vantaggi rispetto al modello di Phong

La sostituzione di $\mathbf{R} \cdot \mathbf{V}$ con $\mathbf{N} \cdot \mathbf{H}$ comporta due vantaggi principali.

Il primo è computazionale: il calcolo di $\mathbf{R}$ richiede una riflessione (equazione 2.2), mentre $\mathbf{H}$ richiede solo una somma e una normalizzazione.

Il secondo riguarda la qualità visiva. Il problema del riflesso speculare troncato descritto nella sezione precedente non si presenta con Blinn-Phong: l'angolo tra **N** e **H** non supera mai i 90° in condizioni normali, eliminando l'artefatto.

Per queste ragioni Blinn-Phong ha sostituito il modello di Phong originale come standard nella grafica in tempo reale.

2.3.4 Limiti del modello

Blinn-Phong risolve il problema del riflesso speculare troncato ma condivide con Phong tutte le altre limitazioni: assenza di conservazione dell'energia, natura empirica dei coefficienti, nessuna distinzione tra materiali metallici e dielettrici.

Queste limitazioni saranno affrontate nel capitolo 3 con l'introduzione del Physically Based Rendering.

2.4 Tecniche di shading

I modelli di illuminazione descritti nelle sezioni precedenti stabiliscono come calcolare il colore di un punto su una superficie, mentre le tecniche di shading stabiliscono in quali punti della geometria tale modello viene valutato e come i risultati ottenuti vengono distribuiti sulla superficie. La scelta della tecnica di shading è indipendente dal modello di illuminazione adottato.

2.4.1 Flat shading

Nel *flat shading* il modello di illuminazione viene valutato una sola volta per faccia, utilizzando la normale del poligono, e il colore risultante viene assegnato uniformemente all'intera faccia senza alcuna interpolazione. Il risultato è un'immagine in cui i singoli poligoni sono chiaramente distinguibili, il che lo rende inadatto alla rappresentazione di superfici curve.

Dal punto di vista implementativo, il flat shading si ottiene assegnando a ogni vertice di un poligono la stessa normale, pari a quella della faccia, senza condividerla con le facce adiacenti.

2.4.2 Gouraud shading

Il *Gouraud shading*, proposto da Henri Gouraud nel 1971 [3], valuta il modello di illuminazione una volta per vertice. Il colore calcolato in ciascun vertice viene poi interpolato sulla superficie del poligono durante la rasterizzazione. Per ottenere transizioni più morbide tra facce adiacenti, la normale associata a ciascun vertice non coincide generalmente con la normale di una singola faccia, ma viene calcolata come media delle normali delle facce che condividono quel vertice. In questo modo si ottiene una variazione più continua dell'illuminazione sulla superficie, riducendo l'aspetto sfaccettato tipico del flat shading.

Rispetto al flat shading, il risultato è una superficie con variazioni di colore continue, che elimina le discontinuità tra facce adiacenti. Tuttavia l'interpolazione del colore presenta un limite significativo: i riflessi speculari vengono calcolati solo nelle posizioni dei vertici. Se un riflesso cade al centro di un poligono, lontano dai vertici, il Gouraud shading non è in grado di catturarlo. Il riflesso risulta quindi assente o fortemente distorto.

2.4.3 Phong shading

Il *Phong shading*, proposto nello stesso articolo del modello di illuminazione [1], risolve i limiti del Gouraud shading con un approccio diverso: anziché interpolare il colore tra i vertici, interpola le normali. Il modello di illuminazione viene poi valutato una volta per frammento, utilizzando la normale interpolata. Poiché l'interpolazione non ne preserva la lunghezza unitaria, è necessario rinormalizzarla prima di procedere con i calcoli dell'illuminazione.

Questo approccio produce riflessi speculari corretti indipendentemente dalla densità della mesh, poiché il calcolo avviene in ogni pixel e non solo nei vertici. Il costo computazionale è più alto (il modello di illuminazione viene valutato per ogni frammento anziché per ogni vertice) ma sulle GPU moderne è una differenza trascurabile.

Tutti gli shader sviluppati in seguito utilizzeranno il Phong shading, poiché è l'unico che produce risultati accurati in presenza di riflessi speculari.

(a) Flat shading **(b)** Gouraud shading **(c)** Phong shading

Figura 2.6: Confronto tra le tre tecniche di shading applicate a una sfera a bassa risoluzione.

Le implementazioni HLSL dei modelli di Phong e Blinn-Phong e delle tre tecniche di shading sono presentate nel capitolo 5.

Capitolo 3

Physically Based Rendering (PBR)

I modelli di Phong e Blinn-Phong, presentati nel capitolo precedente, sono modelli empirici: producono risultati visivamente plausibili, ma non si basano su principi fisici e i loro parametri vengono regolati manualmente, senza vincoli che ne garantiscano la correttezza. Il risultato è che, al variare delle condizioni di illuminazione, gli stessi materiali possono apparire convincenti in una scena e innaturali in un'altra.

Il *Physically Based Rendering* (PBR) nasce per superare queste limitazioni: anziché approssimare l'aspetto visivo con formule empiriche, adotta un approccio fisicamente plausibile all'interazione tra luce e materia, basato su principi come la conservazione dell'energia e la teoria delle microfacce. I materiali vengono descritti tramite grandezze fisicamente interpretabili (rugosità e metallicità), e l'approccio adottato è costruito in modo da rispettare vincoli fisici, tra cui il principio di conservazione dell'energia.

Questo capitolo presenta i principi fisici alla base del PBR, la *Bidirectional Reflectance Distribution Function* (BRDF) di Cook-Torrance e i suoi termini costitutivi, il *workflow metallic/roughness* adottato nei motori moderni e un'introduzione all'*Image-Based Lighting*. L'implementazione HLSL del modello è presentata nel capitolo 5, dedicato alla parte pratica del lavoro.

3.1 Principi fisici

Il PBR si fonda su alcuni principi fisici che ne vincolano il comportamento e lo distinguono dai modelli empirici.

3.1.1 Conservazione dell'energia

Il principio di conservazione dell'energia impone che la quantità totale di luce riflessa da una superficie non possa superare quella della luce incidente. Nei modelli classici questo vincolo non è garantito: i coefficienti k_a, k_d e k_s sono indipendenti tra loro e nulla impedisce che la loro combinazione produca una quantità di luce riflessa superiore a quella ricevuta. Nel PBR, invece, la luce incidente viene suddivisa in una componente riflessa e una che penetra nel materiale e viene diffusa. La formulazione del modello assicura che la somma di questi contributi non superi l'energia disponibile.

Di conseguenza, una superficie molto riflettente avrà necessariamente una componente diffusa debole, e viceversa. Questo produce un comportamento visivamente coerente: i materiali risultano convincenti sotto qualsiasi condizione di illuminazione, senza bisogno di regolazioni manuali.

3.1.2 Teoria delle microfacce

Nel PBR, una superficie è descritta non come un piano perfettamente liscio, ma come un insieme di piccole facce orientate in direzioni diverse, dette *microfacce*, ciascuna delle quali si comporta come uno specchio perfetto riflettendo la luce in un'unica direzione determinata dal proprio orientamento.

L'aspetto del materiale dipende dalla distribuzione degli orientamenti delle microfacce. Una superficie liscia presenta microfacce ben allineate: la luce viene riflessa quasi tutta nella stessa direzione, con un riflesso nitido e concentrato. Una superficie rugosa, invece, ha microfacce orientate caoticamente: la luce viene dispersa in molte direzioni, con un riflesso ampio e sfumato.

(a) Superficie liscia: microfacce allineate **(b)** Superficie rugosa: microfacce non allineate

Figura 3.1: Rappresentazione delle microfacce. Immagini tratte da [4].

Questa distribuzione viene descritta dal parametro *roughness* (rugosità), compreso tra 0 (superficie perfettamente liscia) e 1 (superficie completamente rugosa).

3.1.3 Riflessione e rifrazione

Quando la luce colpisce la superficie di un oggetto, una parte viene riflessa e una parte viene rifratta, penetrando nel materiale. La proporzione tra le due è descritta dalle *equazioni di Fresnel*, trattate nella sezione 3.2.3.

Il comportamento della luce rifratta dipende dal tipo di materiale:

- Nei **dielettrici** (plastica, legno, ceramica) la luce rifratta viene diffusa internamente e riemerge in direzioni casuali, dando origine alla componente diffusa; nei modelli trattati si assume, semplificando, che la luce riemerga dallo stesso punto in cui è penetrata. Il riflesso speculare assume il colore della luce incidente, indipendentemente dal materiale.
- Nei **metalli** (oro, rame, alluminio) la luce rifratta viene assorbita e convertita in calore: la componente diffusa è quindi praticamente assente e tutta la risposta visiva è data dal riflesso speculare, il cui colore dipende sia dalle proprietà del metallo che dalla luce incidente.

Questa distinzione, assente nei modelli classici, viene modellata dal parametro *metallic*.

3.2 La BRDF di Cook-Torrance

Per descrivere come una superficie riflette la luce si utilizza una funzione chiamata BRDF (*Bidirectional Reflectance Distribution Function*). Essa definisce, data una

direzione di illuminazione $\mathbf{L}$ e una direzione di osservazione $\mathbf{V}$, la relazione tra la luce incidente e quella riflessa dalla superficie verso l'osservatore.

La BRDF non fornisce il colore finale di un punto: descrive solo il rapporto tra luce ricevuta e luce riflessa. Per ottenere l'illuminazione effettiva, il contributo di ciascuna sorgente si ottiene moltiplicando la BRDF per la luce incidente L_i e per il coseno dell'angolo di incidenza $\mathbf{N} \cdot \mathbf{L}$; il risultato finale è la somma dei contributi di tutte le sorgenti presenti nella scena:

$$L_o = \sum_{\text{luci}} f_r \cdot L_i \cdot (\mathbf{N} \cdot \mathbf{L}) \tag{3.1}$$

dove L_o è l'intensità della luce riflessa verso l'osservatore, f_r è la BRDF, L_i è l'intensità della luce proveniente dalla direzione $\mathbf{L}$ e $\mathbf{N} \cdot \mathbf{L}$ è il coseno dell'angolo di incidenza, che riduce il contributo delle sorgenti che illuminano la superficie con un angolo radente.

Questa è la forma semplificata della *rendering equation* per sorgenti luminose discrete [5]. La componente ambientale non è inclusa: nel PBR viene gestita dall'Image-Based Lighting (sezione 3.4) o approssimata con un termine costante.

Anche i modelli classici di Phong e Blinn-Phong possono essere considerati BRDF, sebbene empirici. Il PBR utilizza invece BRDF fisicamente accurate: quella più diffusa per il rendering in tempo reale è la BRDF di Cook-Torrance [6], basata sulla teoria delle microfacce introdotta nella sezione 3.1.2.

3.2.1 Struttura della BRDF

La BRDF di Cook-Torrance è composta da un termine diffuso e uno speculare:

$$f_r = f_{\text{diffuse}} + f_{\text{specular}} \tag{3.2}$$

La ripartizione dell'energia tra i due termini è regolata dal termine di Fresnel, come si vedrà nella sezione 3.2.3.

Il termine diffuso f_{diffuse} è dato dal modello di Lambert (lo stesso utilizzato nei modelli classici), che descrive una superficie che riflette la luce uniformemente in tutte le direzioni. Nella sua forma base è dato da:

$$f_{\text{diffuse}} = \frac{c}{\pi} \tag{3.3}$$

dove c è il colore del materiale (*albedo*). La divisione per π è un fattore di normalizzazione che garantisce la conservazione dell'energia. Questo termine verrà completato nella sezione 3.2.3 con il fattore che ne regola il peso rispetto alla componente speculare. A differenza dei modelli classici, dove il coseno dell'angolo tra $\mathbf{N}$ e $\mathbf{L}$ viene incluso direttamente nella formula della componente diffusa, nel PBR esso non fa parte della BRDF ma viene introdotto separatamente nella rendering equation (3.1).

Il termine speculare f_{specular} costituisce il nucleo del modello ed è definito come:

$$f_{\text{specular}} = \frac{D \cdot F \cdot G}{4\,(\mathbf{N} \cdot \mathbf{V})\,(\mathbf{N} \cdot \mathbf{L})} \tag{3.4}$$

dove D, F e G modellano ciascuno un particolare fenomeno fisico:

- D (*normal distribution function*): descrive la distribuzione statistica degli orientamenti delle microfacce.
- F (*Fresnel*): descrive la proporzione di luce riflessa rispetto a quella rifratta in funzione dell'angolo di incidenza.
- G (*geometry function*): modella l'auto-ombreggiatura tra le microfacce.

Il denominatore $4\,(\mathbf{N} \cdot \mathbf{V})\,(\mathbf{N} \cdot \mathbf{L})$ è un fattore correttivo derivato dal modello a microfacce.

Le sottosezioni successive descrivono nel dettaglio ciascuno dei tre termini.

3.2.2 Termine D (normal distribution function)

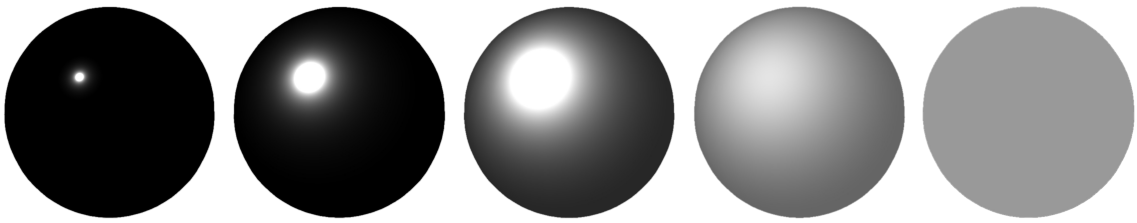
Il termine D è una funzione che descrive la distribuzione statistica degli orientamenti delle microfacce. Data una direzione $\mathbf{H}$ (halfway vector), D descrive la concentrazione di microfacce orientate in quella direzione, ovvero quelle che riflettono effettivamente la luce verso l'osservatore.

L'aspetto della superficie dipende da come queste microfacce sono distribuite. Su una superficie liscia (rugosità bassa) sono quasi tutte allineate alla normale: poche microfacce contribuiscono al riflesso, che risulterà piccolo e intenso. Su una superficie rugosa (rugosità alta) gli orientamenti sono molto più variabili: tante microfacce contribuiscono al riflesso, distribuendolo su un'area ampia ma con minore intensità.

La funzione D utilizzata nel modello di Cook-Torrance è la Trowbridge-Reitz (GGX) [7], definita come:

$$D(\mathbf{N}, \mathbf{H}, \alpha) = \frac{\alpha^2}{\pi \left((\mathbf{N} \cdot \mathbf{H})^2(\alpha^2 - 1) + 1\right)^2} \tag{3.5}$$

dove $\mathbf{N}$ è la normale alla superficie, $\mathbf{H}$ è l'halfway vector (tra la direzione della sorgente luminosa e quella dell'osservatore) e α è un parametro di rugosità derivato dalla *roughness* del materiale. In molte implementazioni, tra cui quella adottata in questo progetto, si pone $\alpha = roughness^2$, poiché produce una variazione percettivamente più lineare al variare del parametro.



(a) 0.05 **(b)** 0.2 **(c)** 0.5 **(d)** 0.8 **(e)** 1.0

Figura 3.2: Visualizzazione del termine D al variare della rugosità. All'aumentare di α la distribuzione delle microfacce diventa più dispersa, producendo un riflesso più ampio e meno intenso.

3.2.3 Termine F (Fresnel)

Il termine F descrive l'effetto Fresnel: la riflettanza di una superficie, ovvero la proporzione di luce che essa riflette, varia in funzione dell'angolo tra la direzione di osservazione $\mathbf{V}$ e la normale $\mathbf{N}$. In particolare, la riflettanza aumenta quanto più la superficie viene osservata di taglio. Questo fenomeno è facilmente osservabile in natura: guardando uno specchio d'acqua dall'alto è spesso possibile vedere il fondo, mentre osservandolo in lontananza prevale il riflesso del paesaggio circostante.

Figura 3.3: Effetto Fresnel su un lago: in primo piano l'acqua appare trasparente, mentre in lontananza riflette il paesaggio.

Le equazioni di Fresnel originali sono complesse e poco pratiche per il calcolo in tempo reale. Nel modello di Cook-Torrance si utilizza l'approssimazione proposta da Schlick [8]:

$$F(\mathbf{H}, \mathbf{V}, F_0) = F_0 + (1 - F_0)(1 - (\mathbf{H} \cdot \mathbf{V}))^5 \tag{3.6}$$

dove $\mathbf{H}$ è l'halfway vector, $\mathbf{V}$ è la direzione dell'osservatore e F_0 è la *riflettanza*

base del materiale, ovvero la frazione di luce riflessa quando l'osservatore guarda la superficie perpendicolarmente. Il valore di F_0 dipende dal tipo di materiale:

- per i dielettrici, F_0 è un valore basso e acromatico, tipicamente intorno a 0.04;
- per i metalli, F_0 è un valore alto e colorato, coincidente con il colore del materiale. È questo che produce il riflesso colorato dei metalli.

Materiale	F_0	**Colore**
Acqua	(0.02, 0.02, 0.02)	
Vetro	(0.03, 0.03, 0.03)	
Plastica	(0.05, 0.05, 0.05)	
Diamante	(0.17, 0.17, 0.17)	
Ferro	(0.56, 0.57, 0.58)	
Rame	(0.95, 0.64, 0.54)	
Oro	(1.00, 0.71, 0.29)	
Alluminio	(0.91, 0.92, 0.92)	
Argento	(0.95, 0.93, 0.88)	

Tabella 3.1: Valori di F_0 per alcuni materiali comuni. Sopra la linea i dielettrici, con valori bassi e acromatici; sotto i metalli, con valori alti e colorati.

Il parametro *metallic* (con valori compresi tra 0 e 1) controlla l'interpolazione di F_0 tra il valore tipico dei dielettrici (0.04) e il colore dell'albedo del materiale, usato come F_0 per i metalli:

$$F_0 = \text{lerp}(0.04, \textit{albedo}, \textit{metallic}) \tag{3.7}$$

Il termine di Fresnel definisce anche la ripartizione dell'energia tra le componenti diffusa e speculare. La frazione di luce destinata alla componente diffusa è data da:

$$k_d = (1 - F)(1 - \text{metallic}) \tag{3.8}$$

Il fattore $(1 - F)$ garantisce la conservazione dell'energia: la porzione di luce non riflessa specularmente viene assegnata alla componente diffusa. Il termine $(1 -$

metallic) annulla invece la componente diffusa nei materiali metallici, i quali riflettono la luce quasi esclusivamente in modo speculare.

Il termine diffuso della BRDF (equazione 3.2) diventa quindi:

$$f_{\text{diffuse}} = k_d \cdot \frac{c}{\pi} \tag{3.9}$$

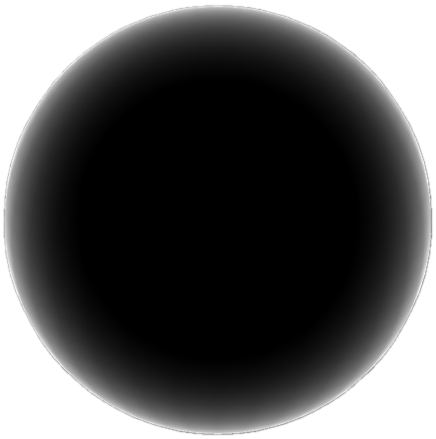
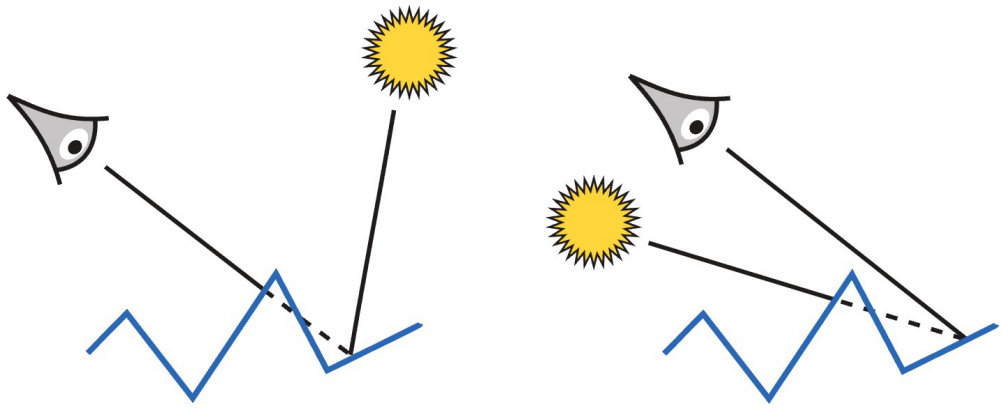


Figura 3.4: Visualizzazione del termine F su una sfera: la riflettanza aumenta alle zone periferiche, dove l'angolo tra $\mathbf{V}$ e $\mathbf{N}$ è maggiore.

3.2.4 Termine G (geometry function)

Il termine G modella il fenomeno di *auto-ombreggiatura* delle microfacce: su una superficie rugosa, alcune microfacce possono oscurare la luce in arrivo su quelle vicine (*shadowing*), mentre altre possono bloccare la luce riflessa prima che raggiunga l'osservatore (*masking*). Entrambi gli effetti riducono la quantità di luce effettivamente visibile e risultano tanto più rilevanti quanto più la superficie è rugosa.



(a) masking (b) shadowing

Figura 3.5: Nel masking (a) la microfaccia, pur orientata correttamente, non è visibile all'osservatore perché coperta da una microfaccia adiacente; nello shadowing (b) la luce non raggiunge la microfaccia perché bloccata da una vicina.

La funzione G combina i contributi di masking e shadowing tramite il *metodo di Smith*, che li valuta separatamente nelle direzioni dell'osservatore e della sorgente luminosa:

$$G(\mathbf{N}, \mathbf{V}, \mathbf{L}, k) = G_1(\mathbf{N}, \mathbf{V}, k) \cdot G_1(\mathbf{N}, \mathbf{L}, k) \tag{3.10}$$

dove $G_1(\mathbf{N}, \mathbf{V}, k)$ modella il masking, ossia la frazione di microfacce visibili all'osservatore, mentre $G_1(\mathbf{N}, \mathbf{L}, k)$ modella lo shadowing, cioè la frazione di microfacce raggiunte dalla luce. Il parametro k è ricavato dalla rugosità ed è definito più avanti.

Il prodotto dei due contributi risulta diverso da zero solo quando una microfaccia è contemporaneamente visibile all'osservatore e illuminata dalla sorgente luminosa.

Per ciascuno dei due fattori si utilizza l'approssimazione di Schlick-GGX:

$$G_1(\mathbf{N}, \mathbf{X}, k) = \frac{\mathbf{N} \cdot \mathbf{X}}{(\mathbf{N} \cdot \mathbf{X})(1 - k) + k} \tag{3.11}$$

dove **X** è un vettore generico sostituito rispettivamente con **V** e **L** per ottenere i due fattori della formula di Smith. Il parametro k è ricavato dalla *roughness* del materiale e dipende dal contesto: nel caso di luce diretta si utilizza:

$$k = \frac{(roughness + 1)^2}{8} \tag{3.12}$$

L'effetto visivo di G è quello di scurire i bordi degli oggetti, dove la superficie è osservata o illuminata ad angoli radenti, in modo più marcato all'aumentare della rugosità.

(a) 0.1 **(b)** 0.5 **(c)** 1.0

Figura 3.6: Visualizzazione del termine G al variare della rugosità. All'aumentare della rugosità, l'auto-ombreggiatura delle microfacce attenua maggiormente le zone illuminate a basso angolo di incidenza.

Figura 3.7: Una sfera renderizzata con il modello di Cook-Torrance.

3.3 Il workflow metallic/roughness

L'approccio PBR descrive i materiali in termini di grandezze fisiche: riflettanza base F_0, rugosità delle microfacce, distinzione tra metalli e dielettrici. Per rendere questi parametri utilizzabili in pratica, i motori di rendering moderni adottano il *workflow metallic/roughness*, che riduce la configurazione di un materiale a pochi parametri intuitivi.

3.3.1 Parametri di un materiale

Un materiale PBR nel workflow metallic/roughness è descritto da tre parametri principali:

- **Albedo**: il colore base del materiale. Per i dielettrici rappresenta il colore della componente diffusa; per i metalli, dove la componente diffusa è assente, viene utilizzato come valore di F_0, determinando il colore del riflesso speculare.
- **Metallic**: un valore compreso tra 0 e 1 che indica se il materiale è un dielettrico (0) o un metallo (1), controllando il valore di F_0 e la presenza della componente

diffusa. Per un dielettrico si assume $F_0 = 0.04$ e la componente diffusa è presente; per un metallo F_0 coincide con l'albedo e la componente diffusa è assente. Questa distinzione viene realizzata tramite l'interpolazione introdotta nell'equazione (3.7), in cui il parametro *metallic* regola la transizione tra il valore tipico dei dielettrici e l'albedo del materiale.

- **Roughness**: la rugosità della superficie, compresa tra 0 (perfettamente liscia) e 1 (molto rugosa). Da essa derivano i parametri che controllano i termini D e G della BRDF.

Rispetto ai coefficienti k_a, k_d, k_s dei modelli classici, questi parametri hanno un significato fisico diretto e producono risultati coerenti senza necessità di regolazione manuale: la struttura del modello impedisce configurazioni che violino il principio di conservazione dell'energia.

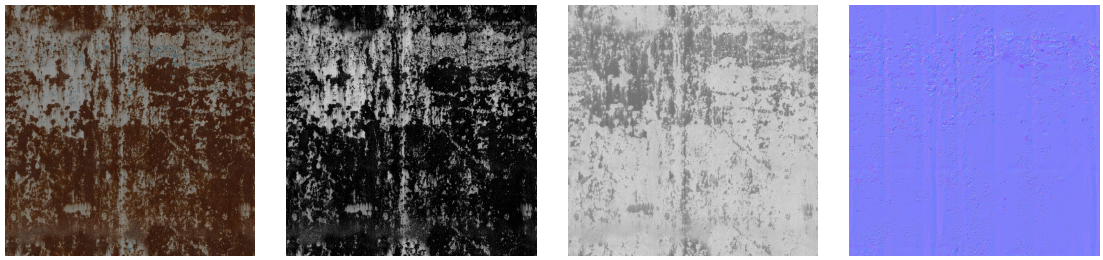
3.3.2 Texture maps

Su un oggetto reale le caratteristiche del materiale variano da punto a punto (metalli arrugginiti, venature del legno, ...). Per rappresentare questa variazione, ogni parametro viene memorizzato in una texture che ne specifica il valore per ciascun pixel della superficie:

- **Albedo map**: definisce il colore base della superficie in ogni punto, indipendentemente dall'illuminazione.
- **Metallic map**: in scala di grigi, indica se un punto della superficie è dielettrico (valore 0) o metallico (valore 1). I valori intermedi sono rari e rappresentano zone di transizione, come bordi o superfici parzialmente ricoperte (ruggine, vernice, sporco).
- **Roughness map**: in scala di grigi, controlla la rugosità della superficie: valori bassi producono riflessi nitidi e concentrati, valori alti riflessi ampi e diffusi.
- **Normal map**: simula dettagli geometrici senza aggiungere geometria reale, fornendo una normale perturbata punto per punto. Ogni pixel codifica quindi

una direzione nello spazio, modificando il modo in cui la luce interagisce con la superficie e dando l'illusione di rilievi e cavità.

Le texture fungono quindi da semplice contenitore per i dati: il modello matematico sottostante rimane invariato, indipendentemente dal fatto che i parametri provengano da una texture o da valori costanti. Nella versione base dello shader Cook-Torrance i parametri del materiale vengono forniti come valori costanti, così da isolare il comportamento del modello di illuminazione. Nel capitolo 5 viene inoltre presentata una variante con texture, che consente di variare albedo, metallic, roughness e orientamento della normale punto per punto sulla superficie.



Albedo Metallic Roughness Normal

Risultato finale

Figura 3.8: Le texture map e il risultato della loro combinazione nel modello di Cook-Torrance.

3.4 Image-Based Lighting

I modelli di illuminazione presentati finora considerano esclusivamente il contributo delle sorgenti luminose presenti nella scena, ovvero la luce diretta. Nella realtà, tuttavia, gli oggetti ricevono illuminazione anche dall'ambiente circostante: ogni superficie della scena contribuisce alla propagazione della luce. Questa componente, detta *luce indiretta*, è quella che nei modelli classici è approssimata mediante un termine ambientale costante.

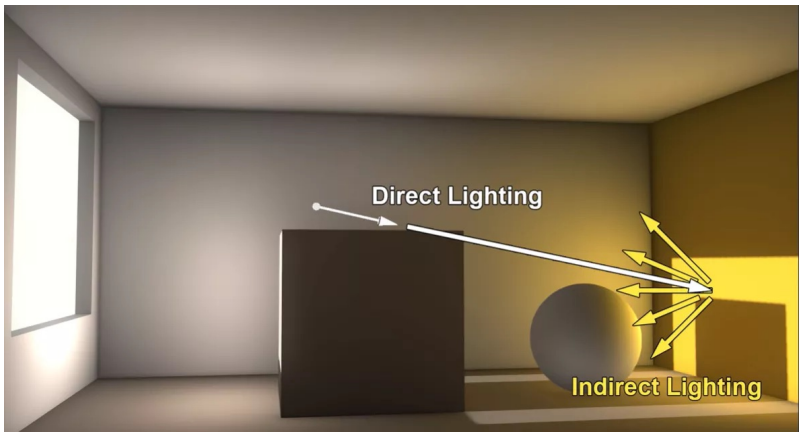


Figura 3.9: Luce diretta e luce indiretta: la luce diretta raggiunge la superficie direttamente dalla sorgente, mentre la luce indiretta è il risultato dei rimbalzi successivi sulle superfici circostanti.

L'*Image-Based Lighting* (IBL) sostituisce questa approssimazione con un approccio più realistico, in cui l'ambiente circostante viene rappresentato tramite una *cubemap*, ossia una collezione di sei texture che descrivono la scena, utilizzata come sorgente luminosa per calcolare l'illuminazione indiretta su ogni punto della superficie.

L'IBL viene generalmente suddiviso in due contributi distinti:

- **Contributo diffuso**: rappresenta la luce ambientale proveniente da tutte le direzioni. Si calcola a partire da una versione pre-elaborata della cubemap, detta *irradiance map*.
- **Contributo speculare**: rappresenta i riflessi dell'ambiente sulle superfici. L'aspetto del riflesso dipende dalla rugosità del materiale: superfici lisce producono riflessi nitidi, mentre superfici rugose producono riflessi più sfocati.

L'implementazione completa dell'IBL esula dagli obiettivi di questa tesi: negli shader sviluppati il contributo indiretto viene approssimato tramite un termine ambientale costante. Per un approfondimento sulla componente diffusa si rimanda a [9], per quella speculare a [10].

Figura 3.10: Una cubemap in formato a croce: le sei facce del cubo vengono "srotolate" in un'unica immagine e rappresentano l'ambiente circostante a 360°.

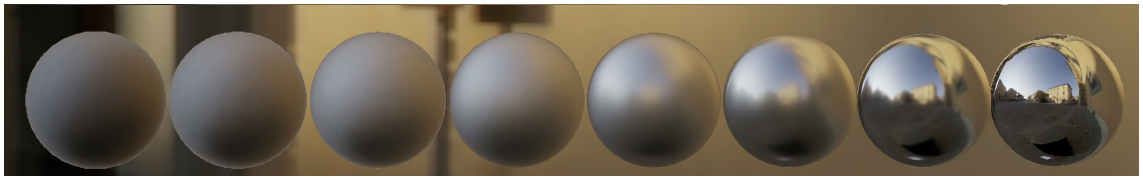


Figura 3.11: Sfere con rugosità crescente illuminate tramite IBL: al diminuire della rugosità i riflessi dell'ambiente diventano progressivamente più nitidi.

Capitolo 4

Shader personalizzati per effetti non fotorealistici

Dopo aver analizzato modelli di illuminazione orientati alla resa realistica, questo capitolo mostra come la stessa pipeline programmabile possa essere utilizzata anche per ottenere effetti stilizzati e non fotorealistici. Gli shader presentati non hanno l'obiettivo di simulare fedelmente il comportamento fisico della luce, ma di evidenziare la flessibilità offerta da vertex e fragment shader nella definizione di modelli di resa personalizzati. Le relative implementazioni HLSL sono presentate nel capitolo 5, dedicato alla parte pratica del lavoro.

4.1 Toon shading

Il *toon shading* è una tecnica di rendering non fotorealistico che imita l'aspetto dei disegni animati e dei fumetti. Anziché produrre sfumature continue come i modelli classici, riduce l'illuminazione a poche fasce di colore con transizioni brusche tra zone illuminate e zone in ombra. Il modello di illuminazione alla base resta quello di Blinn-Phong, con la differenza che i valori calcolati vengono quantizzati in fasce discrete anziché applicati direttamente.

4.1.1 Componente diffusa quantizzata

Mentre nei modelli classici la componente diffusa varia in modo continuo, nel toon shading questo valore viene quantizzato, ovvero suddiviso in un numero discreto di fasce tramite la funzione `floor`:

$$I_{\text{toon}} = \frac{\lfloor (\mathbf{N} \cdot \mathbf{L}) \cdot s \rfloor}{s} \tag{4.1}$$

dove s è il numero di fasce.

4.1.2 Componente speculare con soglia

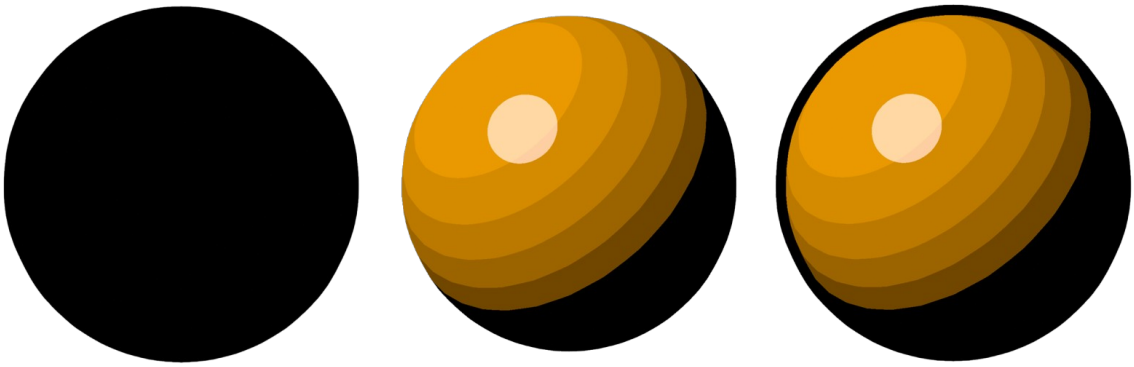
Anche la componente speculare viene stilizzata: anziché sfumare gradualmente con un esponente come nei modelli classici, viene ridotta a una scelta binaria basata su una soglia sul prodotto $\mathbf{N} \cdot \mathbf{H}$:

$$I_s = \begin{cases} k_s & \text{se } \mathbf{N} \cdot \mathbf{H} > t \\ 0 & \text{altrimenti} \end{cases} \tag{4.2}$$

dove t è la soglia e k_s l'intensità del riflesso. Il risultato è un riflesso dai bordi netti, tipico dello stile cartoon.

4.1.3 Linea di contorno

Per completare l'effetto fumetto, è possibile aggiungere un contorno nero attorno all'oggetto. La tecnica utilizzata si basa su un secondo pass di rendering: l'oggetto viene disegnato una prima volta leggermente ingrandito (estrudendo i vertici lungo le normali) e con le facce anteriori nascoste (`Cull Front`), mostrando solo le facce posteriori colorate di nero. Il pass principale disegna poi l'oggetto vero sopra, lasciando visibile il bordo nero come contorno.



(a) Solo contorno (Pass 1) **(b)** Solo shading (Pass 2) **(c)** Risultato finale

Figura 4.1: I due pass del toon shading e il risultato della loro combinazione.

4.2 Hologram shader

Questo shader non si basa su un modello di illuminazione fisico, ma combina tre effetti sovrapposti per simulare l'aspetto di un ologramma: *rim lighting*, pattern a strisce e *flicker*.

4.2.1 Rim lighting

Il *rim lighting* illumina i bordi dell'oggetto sfruttando lo stesso principio dell'effetto Fresnel: il prodotto $\mathbf{N} \cdot \mathbf{V}$ misura quanto la superficie è rivolta verso l'osservatore. Ai bordi, dove la superficie è quasi perpendicolare alla direzione di vista, questo valore tende a zero e il contributo del rim lighting è al massimo:

$$I_{\text{rim}} = (1 - \mathbf{N} \cdot \mathbf{V})^p \tag{4.3}$$

dove p controlla la concentrazione dell'effetto: valori alti producono bordi più sottili e intensi.

4.2.2 Pattern a strisce

Questo effetto consiste in strisce orizzontali che scorrono verso il basso lungo la superficie. Si ottiene applicando una funzione sinusoidale alla coordinata verticale

y del frammento, animata nel tempo:

$$\text{scan} = \begin{cases} 1 & \text{se } \sin(y \cdot d + t \cdot v) > 0 \\ 0 & \text{altrimenti} \end{cases} \tag{4.4}$$

dove d è la densità delle strisce, v la velocità di scorrimento e t il tempo.

4.2.3 Flicker

Il *flicker* simula l'instabilità dell'ologramma modulando l'intensità luminosa con una sinusoide nel tempo:

$$I_{\text{flicker}} = 0.95 + 0.05 \cdot \sin(t \cdot f) \tag{4.5}$$

dove f è la frequenza del tremolio e t il tempo. Il valore oscilla tra 0.90 e 1.00.

4.2.4 Trasparenza

A differenza degli shader precedenti, questo richiede la trasparenza: l'oggetto deve apparire semi-trasparente. In Unity si ottiene abilitando l'*alpha blending*, che combina il colore di ogni frammento con quello già presente nel framebuffer in base al valore alpha, e impostando la coda di rendering a `Transparent`. È inoltre necessario disabilitare la scrittura nel depth buffer, per evitare che le parti trasparenti dell'oggetto nascondano la geometria retrostante.

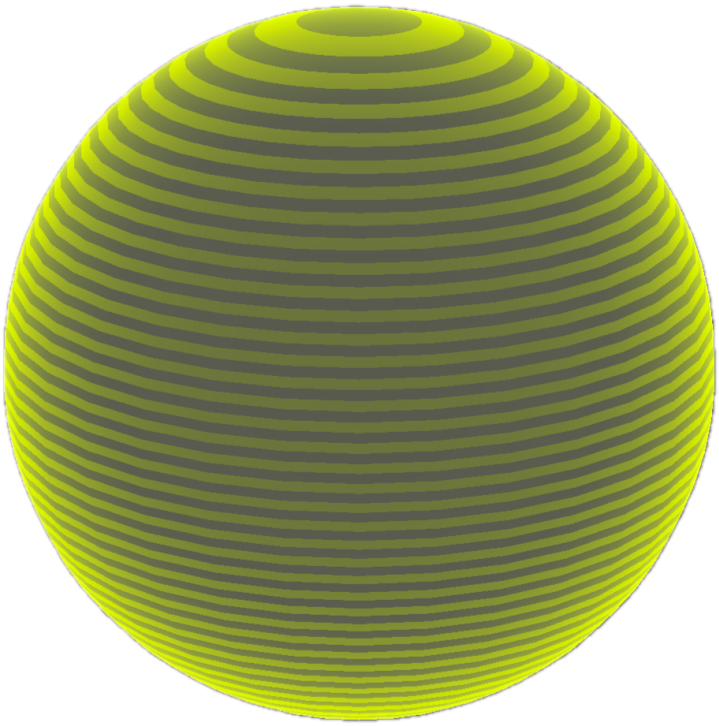


Figura 4.2: Effetto ologramma applicato a un oggetto. I bordi appaiono più luminosi e opachi, le bande orizzontali scorrono verticalmente e un leggero tremolio simula l'instabilità della proiezione.

Gli effetti descritti in questo capitolo, insieme ai modelli di illuminazione classici e all'approccio PBR presentati nei capitoli precedenti, sono stati integrati nel progetto dimostrativo sviluppato in Unity. Il capitolo successivo descrive la struttura del progetto, le scelte implementative adottate e le modalità con cui i diversi shader possono essere confrontati all'interno della scena.

Capitolo 5

Implementazione, test e confronto degli shader in Unity

I capitoli precedenti hanno presentato la teoria dei modelli di illuminazione e delle tecniche di shading. Questo capitolo raccoglie l'implementazione di tutti gli shader in HLSL, la loro integrazione nel progetto Unity, le scene di test realizzate e il confronto visivo e prestazionale tra i modelli.

Tutti gli shader sono stati realizzati come shader personalizzati per la Universal Render Pipeline (URP) di Unity, scritti in HLSL all'interno della struttura ShaderLab introdotta nella sezione 1.4.

5.1 Shader implementati

Sono state implementate le tre tecniche di shading (flat, Gouraud, Phong shading), i tre modelli di illuminazione trattati (Phong, Blinn-Phong, Cook-Torrance), la variante di Cook-Torrance con texture e i due shader non fotorealistici (toon e hologram). La tabella 5.1 ne riassume i parametri principali e lo scopo all'interno del confronto.

Shader	Parametri principali	Scopo nel confronto
Phong	ambient, diffuse, specular, shininess	Mostrare il modello classico originale
Blinn-Phong	ambient, diffuse, specular, shininess	Confrontare la componente speculare con Phong
Cook-Torrance	albedo, metallic, roughness, ambient	Valutare una BRDF fisicamente plausibile
Cook-Torrance con texture	albedo, metallic, roughness e normal map	Mostrare la variazione locale dei parametri
Toon	fasce, soglia speculare, spessore outline	Mostrare una resa non fotorealistica
Hologram	intensità bordi, densità linee, velocità flicker	Mostrare trasparenza ed effetti animati

Tabella 5.1: Riepilogo degli shader implementati, con i parametri principali e lo scopo all'interno del confronto.

5.1.1 Tecniche di shading

Le tre tecniche di shading (sezione 2.4) si distinguono per il punto in cui viene valutato il modello di illuminazione e per i dati scambiati tra vertex e fragment shader. Il modello di illuminazione scelto è indipendente dalla tecnica di shading. Per questo motivo, nei frammenti di codice seguenti, il calcolo dell'illuminazione è lasciato come segnaposto.

Nel **flat shading** il modello di illuminazione viene valutato nel vertex shader e il risultato è annotato con il modificatore `nointerpolation`, che impedisce alla GPU di interpolare il colore tra i vertici: ogni frammento del triangolo riceve così il valore calcolato in un unico vertice, ottenendo un colore uniforme per faccia.

Listing 5.1: Implementazione del flat shading.

```
struct Attributes {
    float4 positionOS : POSITION;
    float3 normalOS : NORMAL;
};

struct Varyings {
    float4 positionCS : SV_POSITION;
    nointerpolation float3 color : TEXCOORD0; // colore non interpolato
};

Varyings vert(Attributes IN) {
    Varyings OUT;
    OUT.positionCS = TransformObjectToHClip(IN.positionOS.xyz);
    float3 posWS = TransformObjectToWorld(IN.positionOS.xyz);
    float3 N = normalize(TransformObjectToWorldNormal(IN.normalOS));
    OUT.color = ... // calcolo dell'illuminazione
    return OUT;
}

float4 frag(Varyings IN) : SV_Target {
    return float4(IN.color, 1.0);
}
```

Nel **Gouraud shading** il calcolo dell'illuminazione avviene sempre nel vertex shader, ma il colore viene interpolato tra i vertici. L'unica differenza rispetto al flat shading è la rimozione di `nointerpolation`:

Listing 5.2: Implementazione del Gouraud shading.

```
struct Attributes {
    float4 positionOS : POSITION;
    float3 normalOS : NORMAL;
};

struct Varyings {
    float4 positionCS : SV_POSITION;
    float3 color  : TEXCOORD0; // colore interpolato
};

Varyings vert(Attributes IN) {
    Varyings OUT;
    OUT.positionCS = TransformObjectToHClip(IN.positionOS.xyz);
    float3 posWS = TransformObjectToWorld(IN.positionOS.xyz);
    float3 N = normalize(TransformObjectToWorldNormal(IN.normalOS));
    OUT.color = ... // calcolo dell'illuminazione
    return OUT;
}

float4 frag(Varyings IN) : SV_Target {
    return float4(IN.color, 1.0);
}
```

Il fragment shader è identico a quello del flat shading: restituisce il colore senza ulteriori calcoli. La differenza visiva è interamente dovuta all'interpolazione operata dalla rasterizzazione.

Nel **Phong shading** il calcolo dell'illuminazione viene spostato nel fragment shader. Il vertex shader non calcola più il colore, ma passa la normale e la posizione, che vengono interpolate dalla rasterizzazione. Il fragment shader utilizza la normale interpolata (rinormalizzata) per valutare il modello di illuminazione in ogni frammento:

Listing 5.3: Implementazione del Phong shading.

```
struct Attributes {
    float4 positionOS : POSITION;
    float3 normalOS : NORMAL;
};

struct Varyings {
    float4 positionCS : SV_POSITION;
    // normale e posizione del frammento, interpolate
    float3 normalWS : TEXCOORD0;
    float3 positionWS : TEXCOORD1;
};

Varyings vert(Attributes IN) {
    Varyings OUT;
    OUT.positionCS = TransformObjectToHClip(IN.positionOS.xyz);
    OUT.normalWS = TransformObjectToWorldNormal(IN.normalOS);
    OUT.positionWS = TransformObjectToWorld(IN.positionOS.xyz);
    return OUT;
}

float4 frag(Varyings IN) : SV_Target {
    float3 N = normalize(IN.normalWS);
    return ... // calcolo dell'illuminazione
}
```

Tutti gli shader presentati nel resto del capitolo utilizzano il Phong shading, poiché produce riflessi speculari accurati indipendentemente dalla densità della mesh.

5.1.2 Struttura comune degli shader

Gli shader che implementano i modelli di illuminazione (Phong, Blinn-Phong e Cook-Torrance) condividono la stessa struttura di base: un vertex shader che trasforma posizione e normale dallo spazio oggetto allo spazio mondo, e un fragment shader che applica il modello di illuminazione per ogni frammento. Le differenze tra uno shader e l'altro riguardano principalmente le proprietà esposte nel blocco `Properties` e il contenuto del fragment shader; il vertex shader e le strutture `Attributes` e `Varyings` restano invariati. Le proprietà del materiale vengono esposte nelle `Properties`, così da poter essere regolate dal pannello descritto nella sezione 5.2.3.

Fa eccezione la variante del modello di Cook-Torrance con texture, descritta nella sezione 5.1.6, che richiede tra gli `Attributes` anche le coordinate UV (per campionare le texture) e la tangente. Dalla tangente e dalla normale, trasformate in world space dal vertex shader, si ricava la bitangente: i tre vettori, trasmessi tramite i `Varyings`, vengono utilizzati nel fragment shader per costruire la matrice **TBN**.

L'esempio seguente mostra la struttura comune con le proprietà dei modelli classici (Phong e Blinn-Phong); il modello di Cook-Torrance adotta la stessa struttura, cambiando solo le proprietà esposte.

Listing 5.4: Struttura comune degli shader per i modelli classici.

```
Shader "Custom/NomeShader" {
   Properties {
      _Color  ("Colore", Color)               = (1, 1, 1, 1)
      _AmbientK ("Ambient", Range(0, 1))      = 0.1
      _DiffuseK ("Diffuse", Range(0, 1))      = 0.7
      _SpecularK("Specular", Range(0, 1))     = 0.5
      _Shininess("Shininess", Range(1, 256)) = 32
   }
   SubShader {
      Tags { "RenderPipeline"="UniversalPipeline" }
      Pass {
         Tags { "LightMode"="UniversalForward" }
         HLSLPROGRAM
         #pragma vertex vert
         #pragma fragment frag
         #include
           "Packages/com.unity.render-pipelines.universal/ShaderLibrary/Core.hlsl"
         #include
           "Packages/com.unity.render-pipelines.universal/ShaderLibrary/Lighting.hlsl"

         struct Attributes {
            float4 positionOS : POSITION;
            float3 normalOS : NORMAL;
         };
         struct Varyings {
            float4 positionCS : SV_POSITION;
            float3 normalWS : TEXCOORD0;
            float3 positionWS : TEXCOORD1;
         };

         float4 _Color;
         float _AmbientK, _DiffuseK, _SpecularK, _Shininess;

         Varyings vert(Attributes IN) {
            Varyings OUT;
            OUT.positionCS = TransformObjectToHClip(IN.positionOS.xyz);
            OUT.normalWS = TransformObjectToWorldNormal(IN.normalOS);
            OUT.positionWS = TransformObjectToWorld(IN.positionOS.xyz);
            return OUT;
         }

         float4 frag(Varyings IN) : SV_Target {
            // fragment shader diverso in base al modello
         }
         ENDHLSL
      }
   }
}
```

5.1.3 Modello di Phong

Lo shader di Phong implementa il modello classico originale (sezione 2.2) e funge da riferimento di partenza nel confronto. Espone i coefficienti del modello (ambient, diffuse e specular), l'esponente di lucentezza (shininess) e il colore del materiale. Il fragment shader traduce l'equazione (2.6), calcolando il vettore di riflessione $\mathbf{R}$ e la componente speculare in funzione dell'angolo tra $\mathbf{R}$ e la direzione di vista $\mathbf{V}$.

Listing 5.5: Fragment shader del modello di Phong.

```
float4 frag(Varyings IN) : SV_Target {
    // Sorgente luminosa
    Light light = GetMainLight();

    // Vettori
    float3 N = normalize(IN.normalWS);
    float3 L = normalize(light.direction);
    float3 V = normalize(GetCameraPositionWS() - IN.positionWS);
    float3 R = reflect(-L, N);

    // Componenti
    float3 ambient = _AmbientK * _Color.rgb;
    float3 diffuse = _DiffuseK * max(dot(N, L), 0.0) * _Color.rgb * light.color;
    float3 specular = _SpecularK * pow(max(dot(R, V), 0.0), _Shininess) * light.color;

    // Risultato finale
    return float4(ambient + diffuse + specular, 1.0);
}
```

5.1.4 Modello di Blinn-Phong

Lo shader di Blinn-Phong (sezione 2.3) differisce da quello di Phong unicamente nel calcolo della componente speculare: il vettore di riflessione **R** viene sostituito dall'halfway vector **H** e la shininess viene moltiplicata per 4 per ottenere un riflesso comparabile.

Listing 5.6: Fragment shader del modello di Blinn-Phong.

```
float4 frag(Varyings IN) : SV_Target {
    // Sorgente luminosa
    Light light = GetMainLight();

    // Vettori
    float3 N = normalize(IN.normalWS);
    float3 L = normalize(light.direction);
    float3 V = normalize(GetCameraPositionWS() - IN.positionWS);
    float3 H = normalize(L + V);

    // Componenti
    float3 ambient = _AmbientK * _Color.rgb;
    float3 diffuse = _DiffuseK * max(dot(N, L), 0.0) * _Color.rgb * light.color;
    float3 specular = _SpecularK * pow(max(dot(N, H), 0.0), _Shininess * 4.0) *
        light.color;

    // Risultato finale
    return float4(ambient + diffuse + specular, 1.0);
}
```

5.1.5 Modello di Cook-Torrance

Lo shader implementa la BRDF di Cook-Torrance (sezione 3.2) e permette di valutare una resa fisicamente plausibile. Adotta la struttura comune descritta nella sezione 5.1.2, esponendo però le proprietà del workflow metallic/roughness (`_Albedo`, `_Metallic`, `_Roughness`) e un termine ambientale (`_AmbientK`) al posto dei coefficienti dei modelli classici.
I tre termini D, F e G della BRDF sono implementati come funzioni HLSL separate, richiamate dal fragment shader.

Listing 5.7: Funzioni dei termini D, F e G della BRDF.

```
// D - Distribuzione delle microfacce
float D_GGX(float3 N, float3 H, float roughness) {
    float a = pow(roughness, 2.0);
    float NdotH = max(dot(N, H), 0.0);
    float denom = pow(NdotH, 2.0) * (pow(a, 2.0) - 1.0) + 1.0;
    return pow(a, 2.0) / (PI * pow(denom, 2.0));
}

// F - Effetto Fresnel
float3 F_Schlick(float3 H, float3 V, float3 F0) {
    float HdotV = max(dot(H, V), 0.0);
    return F0 + (1.0 - F0) * pow(1.0 - HdotV, 5.0);
}

// G - Auto-ombreggiatura
float G1_SchlickGGX(float3 N, float3 X, float roughness) {
    float NdotX = max(dot(N, X), 0.0);
    float k = pow(roughness + 1.0, 2.0) / 8.0;
    return NdotX / (NdotX * (1.0 - k) + k);
}

float G_Smith(float3 N, float3 V, float3 L, float roughness) {
    return G1_SchlickGGX(N, V, roughness) * G1_SchlickGGX(N, L, roughness);
}
```

Il fragment shader assembla i tre termini nella BRDF, combinando la componente speculare con quella diffusa di Lambert e applicando la rendering equation (3.1).

Listing 5.8: Fragment shader del modello di Cook-Torrance.

```
float4 frag(Varyings IN) : SV_Target {
    // Sorgente luminosa
    Light light = GetMainLight();

    // Vettori
    float3 N = normalize(IN.normalWS);
    float3 L = normalize(light.direction);
    float3 V = normalize(GetCameraPositionWS() - IN.positionWS);
    float3 H = normalize(L + V);

    float NdotL = max(dot(N, L), 0.0);
    float NdotV = max(dot(N, V), 0.0);

    // Riflettanza base
    float3 F0 = lerp(0.04, _Albedo.rgb, _Metallic);

    // Termini della BRDF
    float D = D_GGX(N, H, _Roughness);
    float3 F = F_Schlick(H, V, F0);
    float G = G_Smith(N, V, L, _Roughness);

    // Componente diffusa
    float3 kD = (1.0 - F) * (1.0 - _Metallic);
    float3 diffuse = kD * _Albedo.rgb / PI;

    // Componente speculare
    float3 specular = (D * F * G) / (4.0 * NdotV * NdotL + 0.0001);

    // Rendering equation
    float3 color = (diffuse + specular) * light.color * NdotL;

    // Approssimazione del termine ambientale
    color += _AmbientK * _Albedo.rgb;

    return float4(color, 1.0);
}
```

5.1.6 Modello di Cook-Torrance con texture

Questa variante del modello di Cook-Torrance mostra come i parametri del materiale possano variare punto per punto, venendo campionati da texture invece che definiti tramite valori costanti. Espone una albedo map, una metallic map, una roughness map e una normal map. Le funzioni D, F, G e la struttura del fragment shader restano identiche alla versione precedente: cambiano solo la lettura dei parametri e il calcolo della normale.

Listing 5.9: Lettura dei parametri da texture.

```
float3 albedo = SAMPLE_TEXTURE2D(_AlbedoMap, sampler_AlbedoMap, IN.uv).rgb;
float metallic = SAMPLE_TEXTURE2D(_MetallicMap, sampler_MetallicMap, IN.uv).r;
float roughness = SAMPLE_TEXTURE2D(_RoughnessMap, sampler_RoughnessMap, IN.uv).r;
```

La normal map richiede un passaggio aggiuntivo. Le normali memorizzate nella texture sono espresse in un sistema di coordinate locale alla superficie (*tangent space*) e devono essere trasformate in world space per essere utilizzate nei calcoli di illuminazione. Per farlo si costruisce la matrice **TBN** a partire da tre vettori che descrivono l'orientamento della superficie nel mondo: la tangente **T** e la normale **N**, trasformate in world space dal vertex shader, e la bitangente **B**, da esse derivata tramite prodotto vettoriale. Moltiplicando la normale letta dalla texture per questa matrice si ottiene una normale espressa nel sistema di riferimento globale:

Listing 5.10: Applicazione della normal map.

```
// Lettura della normale dalla texture (in tangent space)
float3 normalTS
      = UnpackNormal(SAMPLE_TEXTURE2D(_NormalMap, sampler_NormalMap, IN.uv));

// Matrice TBN: trasforma da tangent space a world space
float3x3 TBN = float3x3(
   normalize(IN.tangentWS),
   normalize(IN.bitangentWS),
   normalize(IN.normalWS));

// Normale perturbata in world space
float3 N = normalize(mul(normalTS, TBN));
```

La normale **N** risultante sostituisce quella interpolata dal vertex shader, dando l'illusione di dettagli geometrici senza aggiungere geometria reale.

5.1.7 Toon shading

Lo shader toon, descritto nella sezione 4.1, produce una resa non fotorealistica quantizzando l'illuminazione di Blinn-Phong. È composto da due pass: il primo disegna il contorno, il secondo applica lo shading stilizzato. Espone il numero di fasce, la soglia speculare e i parametri del contorno.
Il primo pass estrude i vertici lungo la normale e, con `Cull Front`, disegna solo le facce posteriori colorate, creando il bordo.

Listing 5.11: Primo pass del toon shading: linea di contorno.

```
Pass {
   Cull Front // Nasconde le facce anteriori

   // ...

   Varyings vert(Attributes IN) {
      Varyings OUT;
      float3 pos = IN.positionOS.xyz + IN.normalOS * _OutlineSize;
      OUT.positionCS = TransformObjectToHClip(pos);
      return OUT;
   }
   float4 frag(Varyings IN) : SV_Target {
      return _OutlineColor;
   }
}
```

Il secondo pass quantizza la componente diffusa con `floor` e applica una soglia alla speculare.

Listing 5.12: Secondo pass del toon shading: shading quantizzato.

```
float4 frag(Varyings IN) : SV_Target {
   // ... sorgente luminosa e vettori N, L, V, H

   // Componente diffusa quantizzata
   float NdotL = max(dot(N, L), 0.0);
   float toon = floor(NdotL * _Steps) / _Steps;

   // Componente speculare con soglia
   float NdotH = max(dot(N, H), 0.0);
   float spec = NdotH > _SpecThresh ? _SpecularK : 0.0;

   // Colore finale
   float3 color = _Color.rgb * light.color * toon + spec * light.color;
   return float4(color, 1.0);
}
```

5.1.8 Hologram shader

Lo shader hologram, descritto nella sezione 4.2, combina *rim lighting*, pattern a strisce e *flicker*, introducendo inoltre la trasparenza. La configurazione del *pass* abilita l'*alpha blending*, imposta la coda di rendering a `Transparent` e disabilita la scrittura nel depth buffer:

Listing 5.13: Configurazione del pass per la trasparenza.

```
Tags {
    "RenderPipeline"="UniversalPipeline"
    "Queue"="Transparent"
    "RenderType"="Transparent"
}
Blend SrcAlpha OneMinusSrcAlpha
ZWrite Off
```

Il fragment shader combina i tre effetti e calcola l'opacità finale.

Listing 5.14: Fragment shader dell'hologram shader.

```
float4 frag(Varyings IN) : SV_Target {
    float3 N = normalize(IN.normalWS);
    float3 V = normalize(GetCameraPositionWS() - IN.positionWS);

    // Rim lighting
    float rim = pow(1.0 - max(dot(N, V), 0.0), _RimPower);

    // Pattern a strisce
    float stripe = sin(IN.positionWS.y * _StripeDensity + _Time.y * _StripeSpeed);
    stripe = step(0.0, stripe);

    // Flicker
    float flicker = 0.95 + 0.05 * sin(_Time.y * _FlickerSpeed);

    // Opacita'
    float alpha = (rim + _StripeAlpha * stripe) * flicker;
    alpha = clamp(alpha, 0.1, 1.0);

    // Colore finale
    float3 color = _Color.rgb * (rim + 0.3) * flicker;
    return float4(color, alpha);
}
```

5.2 Integrazione nel progetto Unity

Gli shader descritti nella sezione precedente sono stati integrati in un progetto Unity organizzato in più scene: una scena principale interattiva, pensata per esplorare i materiali e modificarne i parametri, e un insieme di scene di test dedicate al confronto tra i diversi modelli. Le scene verranno descritte nel dettaglio nella sezione 5.3; la presente sezione si concentra sugli script che gestiscono il funzionamento del progetto, in particolare la navigazione in prima persona, la selezione degli oggetti, il pannello di controllo e il cambio di scena.

5.2.1 Navigazione in prima persona

La navigazione avviene in prima persona: il movimento è controllato dai tasti WASD e l'orientamento della visuale dal mouse. Il controllo è implementato nello script `FirstPersonCamera`, applicato alla camera. La rotazione verticale (*pitch*) viene applicata alla camera e limitata a $\pm 90°$ per evitare ribaltamenti, mentre la rotazione orizzontale (*yaw*) viene applicata all'oggetto che la contiene. Il movimento utilizza un componente `CharacterController`, che gestisce automaticamente le collisioni con la geometria della scena.

Listing 5.15: Logica di movimento e rotazione della camera.

```
// Rotazione della visuale con il mouse
float mouseX = Input.GetAxis("Mouse X") * lookSpeed;
float mouseY = Input.GetAxis("Mouse Y") * lookSpeed;
rotX = Mathf.Clamp(rotX - mouseY, -90f, 90f);
transform.localRotation = Quaternion.Euler(rotX, 0f, 0f);
transform.parent.Rotate(Vector3.up * mouseX);

// Movimento con i tasti WASD
float h = Input.GetAxis("Horizontal");
float v = Input.GetAxis("Vertical");
Vector3 move = transform.parent.right * h + transform.parent.forward * v;
controller.Move(move * moveSpeed * Time.deltaTime);
```

5.2.2 Selezione degli oggetti

Ogni oggetto modificabile è dotato dello script `SelectableObject`, che ne definisce il nome e l'insieme di materiali disponibili. All'avvio lo script crea una copia di ciascun materiale, in modo che le modifiche apportate durante l'esecuzione siano indipendenti per ogni oggetto e non alterino i materiali originali del progetto. Il primo materiale dell'elenco viene applicato come predefinito.
Poiché i modelli importati possono essere composti da più mesh annidate, lo script raccoglie tutti i `Renderer` presenti nell'oggetto e nei suoi figli, applicando il materiale a ciascuno di essi.

Listing 5.16: Creazione delle istanze dei materiali e applicazione.

```
void Awake() {
   // Renderer dell'oggetto e dei suoi figli
   renderers = GetComponentsInChildren<Renderer>();
   // Copia indipendente di ciascun materiale
   instances = new Material[materials.Length];
   for (int i = 0; i < materials.Length; i++)
      instances[i] = new Material(materials[i]);
   // Applica il materiale predefinito
   if (instances.Length > 0)
      ApplyMaterial(instances[0]);
}

public void ApplyMaterial(Material mat) {
   foreach (var r in renderers)
      r.material = mat;
}
```

La selezione avviene tramite *raycasting*: al click del mouse viene lanciato un raggio dal centro della visuale, e il primo oggetto colpito dotato di `SelectableObject` viene selezionato. Un mirino al centro dello schermo indica il punto di mira.

Listing 5.17: Selezione di un oggetto tramite raycast.

```
Ray ray =
      Camera.main.ScreenPointToRay(new Vector3(Screen.width / 2, Screen.height / 2));
RaycastHit hit;
if (Physics.Raycast(ray, out hit, selectDistance)) {
   var obj = hit.collider.GetComponentInParent<SelectableObject>();
   if (obj != null)
      selected = obj; // apertura del pannello
}
```

5.2.3 Pannello di controllo

Una volta selezionato un oggetto, si apre un pannello che permette di modificarne shader e parametri. Il pannello è realizzato con il sistema *IMGUI* di Unity, scelto per la sua semplicità: non richiede la costruzione di un'interfaccia nell'editor e permette di generare dinamicamente i controlli in base allo shader selezionato.

Il pannello mostra il nome dell'oggetto selezionato, l'elenco degli shader disponibili, i parametri del materiale corrente e i parametri della luce direzionale (intensità, direzione e colore). I controlli dei parametri vengono generati dinamicamente tramite il metodo `HasProperty`, che verifica la presenza di una proprietà nello shader prima di mostrare il relativo controllo:

Listing 5.18: Generazione dinamica dei controlli in base allo shader.

```
1 if (mat.HasProperty("_Metallic"))
2    mat.SetFloat("_Metallic",
3       Slider("Metallic", mat.GetFloat("_Metallic"), 0, 1));
4
5 if (mat.HasProperty("_Roughness"))
6    mat.SetFloat("_Roughness",
7       Slider("Roughness", mat.GetFloat("_Roughness"), 0.05f, 1));
```

In questo modo lo stesso pannello si adatta automaticamente a qualsiasi shader: un materiale Blinn-Phong mostra i controlli per ambient, diffuse, specular e shininess, mentre un materiale PBR mostra metallic e roughness.

Figura 5.1: Il pannello di controllo aperto su un oggetto selezionato, con i controlli generati in base allo shader corrente.

5.2.4 Cambio di scena

Per passare rapidamente da una scena all'altra durante l'esecuzione, senza tornare all'editor, è stato realizzato lo script `SceneSwitcher`, che associa a ciascun tasto numerico il caricamento della scena corrispondente tramite `SceneManager`:

Listing 5.19: Cambio di scena tramite i tasti numerici.

```
void Update() {
    if (Input.GetKeyDown(KeyCode.Alpha1)) SceneManager.LoadScene(0);
    if (Input.GetKeyDown(KeyCode.Alpha2)) SceneManager.LoadScene(1);
    if (Input.GetKeyDown(KeyCode.Alpha3)) SceneManager.LoadScene(2);
    if (Input.GetKeyDown(KeyCode.Alpha4)) SceneManager.LoadScene(3);
    if (Input.GetKeyDown(KeyCode.Alpha5)) SceneManager.LoadScene(4);
    if (Input.GetKeyDown(KeyCode.Alpha6)) SceneManager.LoadScene(5);
}
```

Lo script è presente in tutte le scene, così che il passaggio sia sempre possibile. Le scene devono essere registrate nell'ordine corrispondente nelle impostazioni di build di Unity.

5.3 Scene di test

Oltre alla scena principale interattiva, sono state realizzate alcune scene di test dedicate, pensate per confrontare i modelli in condizioni controllate e riproducibili. Come la scena principale, anche queste permettono la navigazione in prima persona, così da poter osservare gli oggetti da angolazioni diverse; rispetto ad essa, però, hanno una disposizione regolare degli oggetti e un'illuminazione predefinita, in modo che ogni confronto isoli un singolo aspetto. Il passaggio tra le scene avviene con i tasti numerici, come descritto nella sezione 5.2.4.

Per rendere il confronto tra gli shader il più possibile controllato, tutte le scene di test sono state realizzate mantenendo costanti geometria, posizione della camera e configurazione della luce, variando esclusivamente lo shader applicato o i parametri del materiale considerati nel singolo esperimento.

5.3.1 Scena interattiva

È la scena principale del progetto: un ambiente esplorabile in cui l'utente si muove in prima persona, seleziona un oggetto inquadrandolo e ne modifica shader e parametri osservando l'effetto in tempo reale.
L'illuminazione è affidata a un'unica luce direzionale. Limitarsi a una sola sorgente, di cui si controllano direzione, intensità e colore, permette di isolare il comportamento di ciascun modello senza che contributi multipli si sovrappongano, rendendo il confronto più leggibile.
Gli oggetti selezionabili sono impostati in modo che ciascuno parta con un materiale predefinito, ma possa essere riconfigurato con uno qualsiasi degli shader implementati.

Figura 5.2: Panoramica della scena interattiva.

(a) Phong

(b) Blinn-Phong

(c) PBR

(d) PBR con texture

(e) Toon

(f) Hologram

Figura 5.3: Lo stesso oggetto con i sei shader implementati, modificati attraverso il pannello.

5.3.2 Scena panoramica degli shader

Questa scena dispone una serie di sfere identiche, ciascuna associata a uno degli shader implementati: Phong, Blinn-Phong, Cook-Torrance, toon e hologram. Le sfere sono illuminate dalla stessa luce e osservate dallo stesso punto di vista, così da evidenziare a colpo d'occhio le differenze visive complessive tra i modelli, prima dei confronti mirati presentati nelle scene successive.

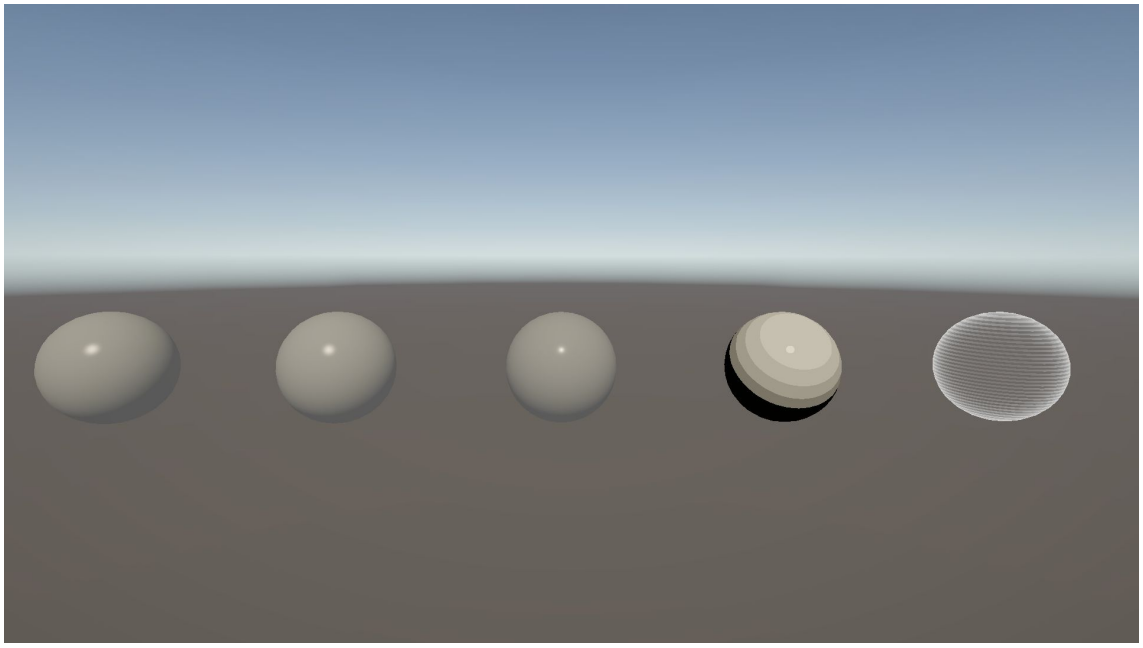


Figura 5.4: Una sfera per ciascuno shader implementato, nella stessa scena e con la stessa illuminazione.

5.3.3 Scena del riflesso speculare troncato

Questa scena è dedicata al confronto tra Phong e Blinn-Phong in presenza del riflesso speculare troncato, descritto nella sezione 2.3.1. Contiene sfere e piani illuminati da una luce radente, quasi parallela alla superficie, con un valore di shininess molto basso: in queste condizioni il taglio netto del riflesso di Phong diventa evidente, mentre Blinn-Phong mantiene una transizione continua. L'analisi dei risultati è riportata nella sezione 5.4.

5.3.4 Scena di confronto tra Blinn-Phong e Cook-Torrance

Questa scena confronta il modello empirico di Blinn-Phong con il modello di Cook-Torrance, su sfere identiche e sotto la stessa illuminazione. Sono previste tre configurazioni, pensate per evidenziare tre differenze distinte:

- un materiale metallico: mette in luce come Blinn-Phong produca un riflesso speculare bianco, mentre Cook-Torrance lo colori in base all'albedo del metallo;
- un materiale con rugosità elevata: evidenzia lo scurimento dei bordi prodotto dal termine G di Cook-Torrance, assente in Blinn-Phong;
- una coppia di sfere lisce osservate da angolazioni diverse: evidenzia l'effetto Fresnel, che in Cook-Torrance intensifica il riflesso speculare agli angoli radenti.

I risultati e il relativo commento sono riportati nella sezione 5.4.

5.3.5 Scena di confronto metallic/roughness

Questa scena dispone una griglia di sfere con shader Cook-Torrance, in cui il parametro metallic varia lungo un asse e la roughness lungo l'altro. Permette di osservare in un'unica immagine come i due parametri modifichino l'aspetto del materiale.
Per evitare di configurare manualmente le decine di sfere della griglia, queste vengono generate durante l'esecuzione da uno script che ne imposta i parametri in base alla posizione:

Listing 5.20: Generazione procedurale della griglia metallic/roughness.

```
for (int x = 0; x < size; x++)
   for (int y = 0; y < size; y++) {
      Vector3 pos = new Vector3(x * spacing, y * spacing, 0);
      GameObject s = Instantiate(spherePrefab, pos, Quaternion.identity);
      Material m = s.GetComponent<Renderer>().material;
      m.SetFloat("_Metallic", x / (float)(size - 1));
      m.SetFloat("_Roughness", Mathf.Max(y / (float)(size - 1), 0.05f));
   }
```

Il valore di roughness viene limitato a un minimo di 0.05: con roughness nulla il termine D della BRDF (sezione 3.2.2) degenera in un riflesso speculare puntiforme, che produce artefatti visivi.
Il risultato e il commento sono riportati nella sezione 5.4.

5.3.6 Scena dei materiali con texture

Questa scena raccoglie alcune sfere con shader Cook-Torrance configurate con set di texture diversi (albedo, metallic, roughness e normal map), corrispondenti a materiali come legno, metallo e pietra. A differenza delle scene precedenti, in cui i parametri del materiale sono definiti come valori costanti e variano solo da una sfera all'altra, in questo caso ogni texture consente di modificare localmente le proprietà del materiale sulla superficie dell'oggetto. La scena permette quindi di evidenziare il ruolo delle texture map nella resa di materiali più complessi e visivamente dettagliati. Il risultato è discusso nella sezione 5.4.4.

5.4 Confronto visivo

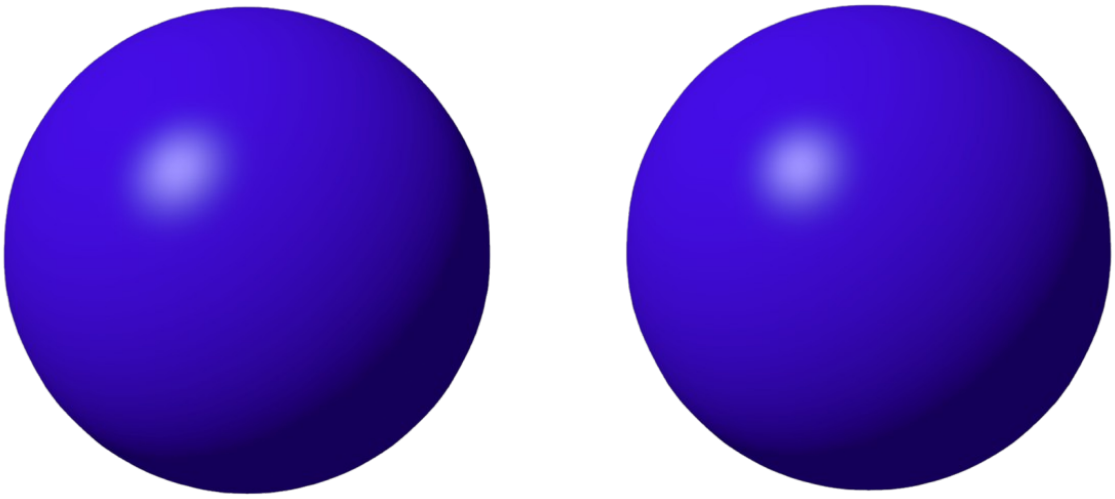
In questa sezione vengono messi a confronto i risultati prodotti dai diversi modelli, a partire dalle scene di test descritte in precedenza. Per ogni confronto, oltre alle immagini, viene fornito un commento che spiega l'origine delle differenze osservate, collegandole alla teoria dei capitoli precedenti.

Confronto	**Scena di riferimento**	**Aspetto analizzato**
Phong e Blinn-Phong	Scena del riflesso speculare troncato	Differenza nella componente speculare
Blinn-Phong e Cook-Torrance	Scena di confronto Blinn-Phong / Cook-Torrance	Differenza tra un modello empirico e uno fisicamente plausibile
Variazione di metallic e roughness	Scena metallic/roughness	Effetto dei parametri del workflow metallic/roughness
Effetto delle texture	Scena dei materiali con texture	Variazione locale dei parametri
Shader non fotorealistici	Scena panoramica	Differenze stilistiche rispetto ai modelli realistici

Tabella 5.2: Corrispondenza tra ogni confronto della sezione 5.4 e la scena di test (sezione 5.3) da cui deriva.

5.4.1 Phong e Blinn-Phong

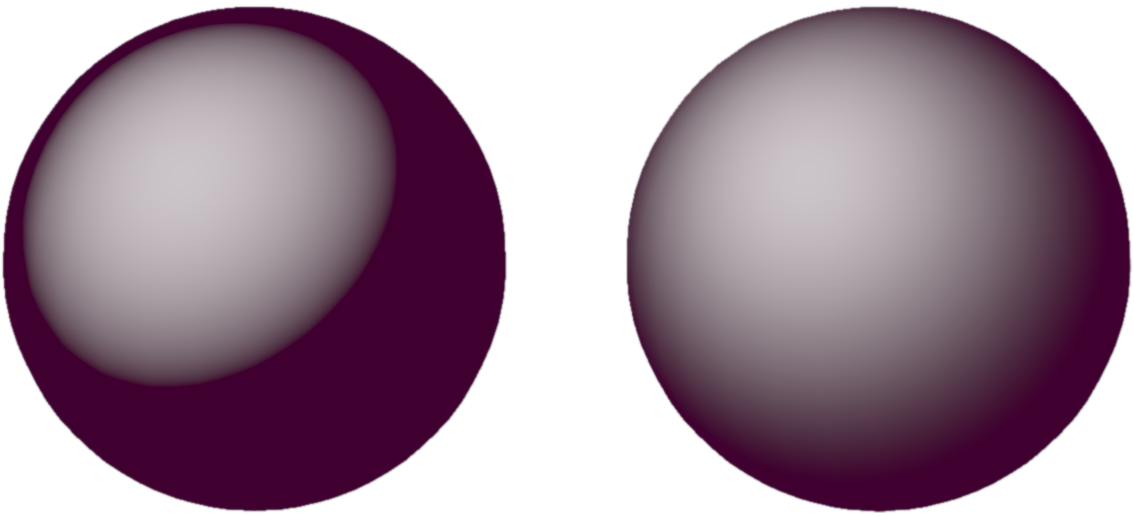
Questo confronto è stato ottenuto nella scena descritta nella sezione 5.3.3, progettata per evidenziare il comportamento della componente speculare nei modelli di Phong e Blinn-Phong. In condizioni di illuminazione normali, su una superficie curva, i modelli di Phong e Blinn-Phong producono risultati molto simili, a patto di equiparare i parametri (in particolare usando per Blinn-Phong un esponente di shininess maggiore, come discusso nella sezione 2.3.1). La differenza nella forma del riflesso speculare è in questo caso sottile e difficilmente percepibile (figura 5.5).



(a) Phong **(b)** Blinn-Phong

Figura 5.5: Confronto tra Phong e Blinn-Phong con parametri equiparati. In condizioni normali i due modelli sono quasi indistinguibili.
Parametri: ambient = 0.04, diffuse = 0.7, specular = 0.5, shininess = 64.

La differenza diventa invece netta in presenza di luce radente e shininess bassa. Nel modello di Phong il riflesso si interrompe bruscamente con un bordo netto, dove l'angolo tra il vettore di riflessione $\mathbf{R}$ e la direzione di vista $\mathbf{V}$ supera i 90° e il prodotto scalare viene azzerato. In Blinn-Phong, dove la speculare dipende dall'angolo tra $\mathbf{N}$ e l'halfway vector $\mathbf{H}$, questa condizione non si verifica e il riflesso sfuma in modo continuo, come mostrato nelle figure 5.6 e 5.7.



(a) Phong **(b)** Blinn-Phong

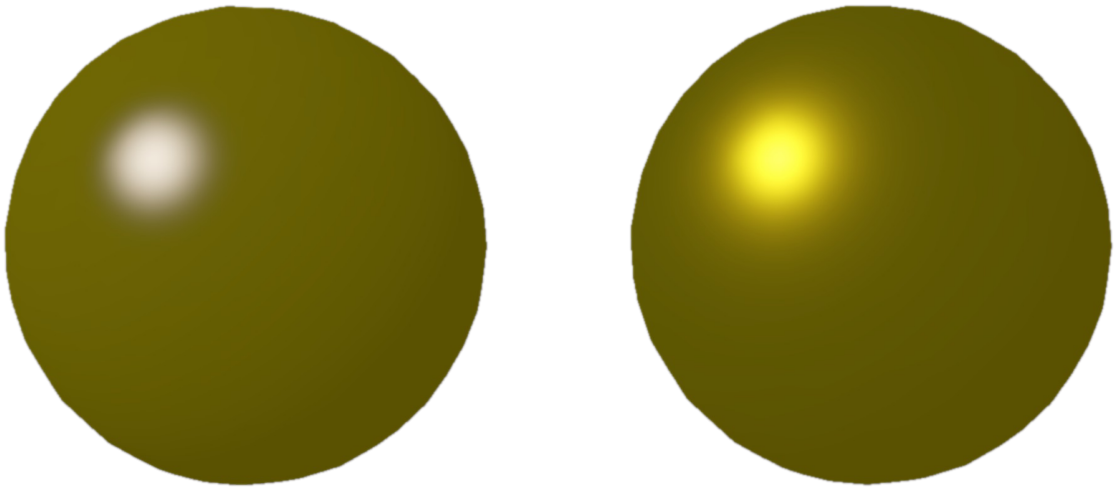
Figura 5.6: Riflesso speculare su una sfera con shininess molto bassa. In Phong il riflesso presenta un taglio netto, assente in Blinn-Phong.
Parametri: ambient = 0.01, diffuse = 0, specular = 1.0, shininess = 1.

(a) Phong **(b)** Blinn-Phong

Figura 5.7: Lo stesso confronto su un piano, visto dall'alto con la sorgente luminosa alle spalle dell'osservatore. Il taglio netto del riflesso di Phong è ancora più evidente che sulla sfera.
Parametri: ambient = 0.01, diffuse = 0, specular = 1.0, shininess = 1.

5.4.2 Blinn-Phong e Cook-Torrance

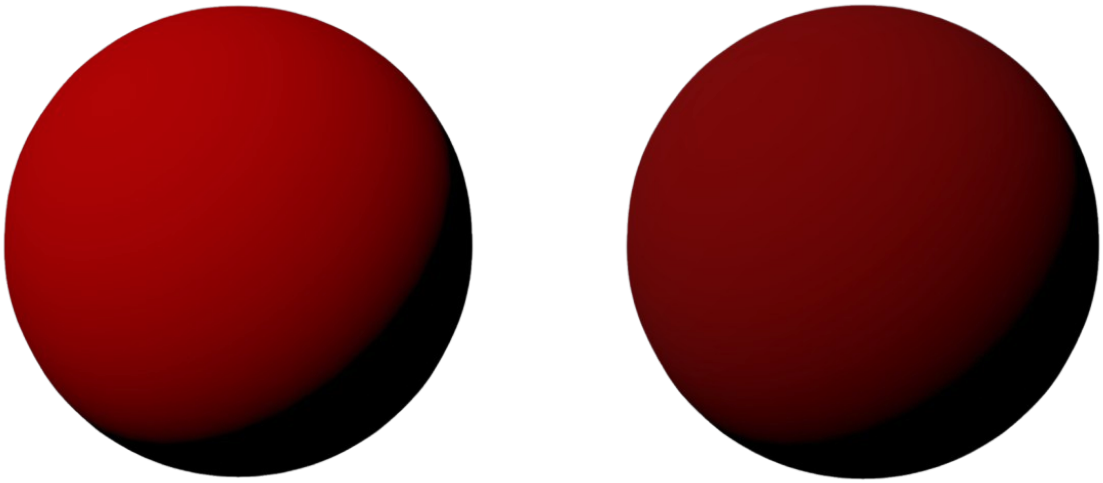
Questo confronto è stato ottenuto nella scena descritta nella sezione 5.3.4, progettata per osservare il comportamento dei due modelli su materiali metallici e rugosi. Il confronto tra Blinn-Phong e Cook-Torrance mette in evidenza il passaggio da un modello empirico a una BRDF fisicamente plausibile. La differenza più immediata emerge nel caso dei materiali metallici. Nel modello Blinn-Phong, la componente speculare assume sempre il colore della sorgente luminosa, indipendentemente dalle proprietà del materiale. Di conseguenza, una sfera dorata presenta un riflesso bianco, percettivamente poco realistico. Nel modello Cook-Torrance, invece, per i materiali metallici la riflettanza base F_0 coincide con l'albedo, come discusso nella sezione 3.2.3; il riflesso speculare assume quindi il colore caratteristico del metallo, producendo una resa visiva più coerente, come mostrato in figura 5.8.



(a) Blinn-Phong **(b)** Cook-Torrance

Figura 5.8: Confronto su un materiale metallico (oro). In Blinn-Phong il riflesso speculare è bianco; in Cook-Torrance assume il colore dell'albedo.
Parametri Blinn-Phong: ambient = 0.1, diffuse = 0.05, specular = 0.75, shininess = 25.
Parametri Cook-Torrance: ambient = 0.08, metallic = 0.7, roughness = 0.3.

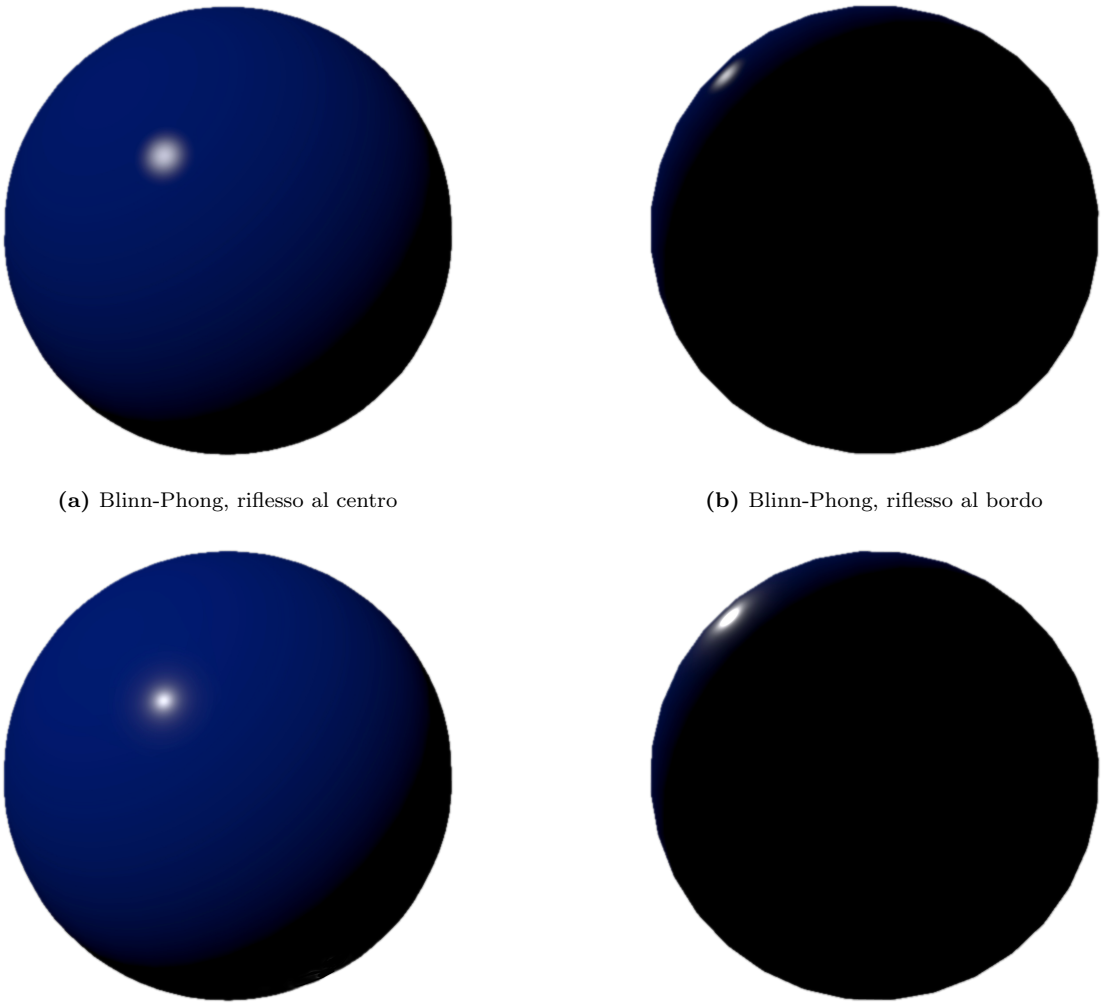
Una seconda differenza emerge con materiali rugosi. Il termine G di Cook-Torrance modella l'auto-ombreggiatura tra le microfacce (descritta nella sezione 3.2.4) attenuando la luce nelle zone osservate o illuminate ad angoli radenti. Il risultato è uno scurimento dei bordi che aumenta con la rugosità, assente in Blinn-Phong, dove la superficie appare uniformemente illuminata.



(a) Blinn-Phong **(b)** Cook-Torrance

Figura 5.9: Confronto con rugosità elevata. In Cook-Torrance i bordi risultano più scuri per effetto del termine G, assente in Blinn-Phong.
Parametri Blinn-Phong: ambient = 0, diffuse = 0.7, specular = 0.
Parametri Cook-Torrance: ambient = 0, metallic = 0, roughness = 1.0.

Una terza differenza riguarda l'effetto Fresnel, presente in Cook-Torrance tramite il termine F (descritto nella sezione 3.2.3) e assente in Blinn-Phong. Poiché negli shader sviluppati non è implementato l'Image-Based Lighting, il Fresnel non si manifesta come riflesso dell'ambiente, ma come aumento della riflettanza speculare agli angoli radenti. Lo si può osservare spostando il punto di vista in modo che il riflesso speculare si avvicini al bordo della sfera: in Cook-Torrance diventa più intenso, mentre in Blinn-Phong resta invariato (figura 5.10).



(a) Blinn-Phong, riflesso al centro

(b) Blinn-Phong, riflesso al bordo

(c) Cook-Torrance, riflesso al centro

(d) Cook-Torrance, riflesso al bordo

Figura 5.10: Effetto Fresnel con luce diretta. In Blinn-Phong il riflesso ha la stessa intensità al centro e al bordo. In Cook-Torrance il riflesso al bordo è più intenso che al centro.
Parametri Blinn-Phong: ambient = 0, diffuse = 0.25, specular = 1.0, shininess = 160.
Parametri Cook-Torrance: ambient = 0, metallic = 0, roughness = 0.2.

Al di là delle singole differenze, il vantaggio principale di Cook-Torrance è la coerenza: grazie alla conservazione dell'energia e ai parametri fisicamente interpretabili, un materiale configurato una volta appare convincente sotto qualsiasi illuminazione, senza bisogno di regolazioni manuali. In Blinn-Phong, invece, i coefficienti vanno spesso ritarati al variare delle condizioni di luce per ottenere un risultato accurato.

5.4.3 Variazione di metallic e roughness

La griglia di sfere generata nella scena descritta nella sezione 5.3.5 mostra l'effetto combinato dei due parametri del workflow metallic/roughness. Lungo l'asse del metallic, all'aumentare del valore la componente diffusa scompare progressivamente e il riflesso speculare assume il colore del materiale: la sfera passa da un aspetto plastico a uno metallico. Lungo l'asse della roughness, all'aumentare del valore il riflesso si allarga e perde intensità, passando da una superficie a specchio a una opaca, come mostrato in figura 5.11.

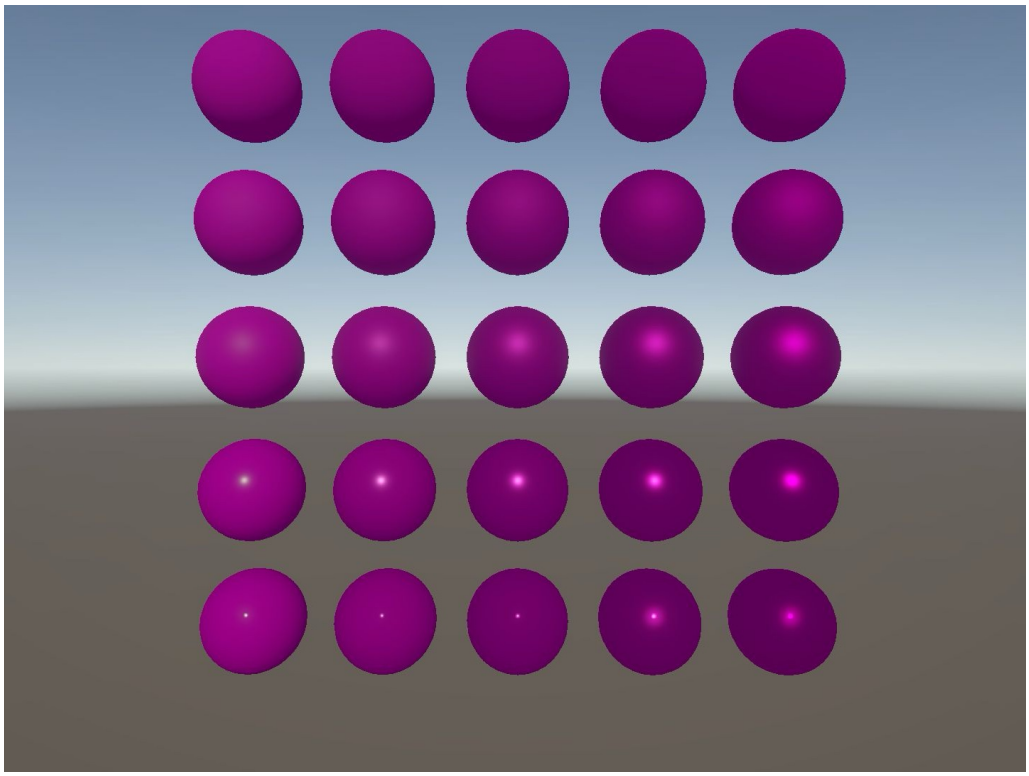
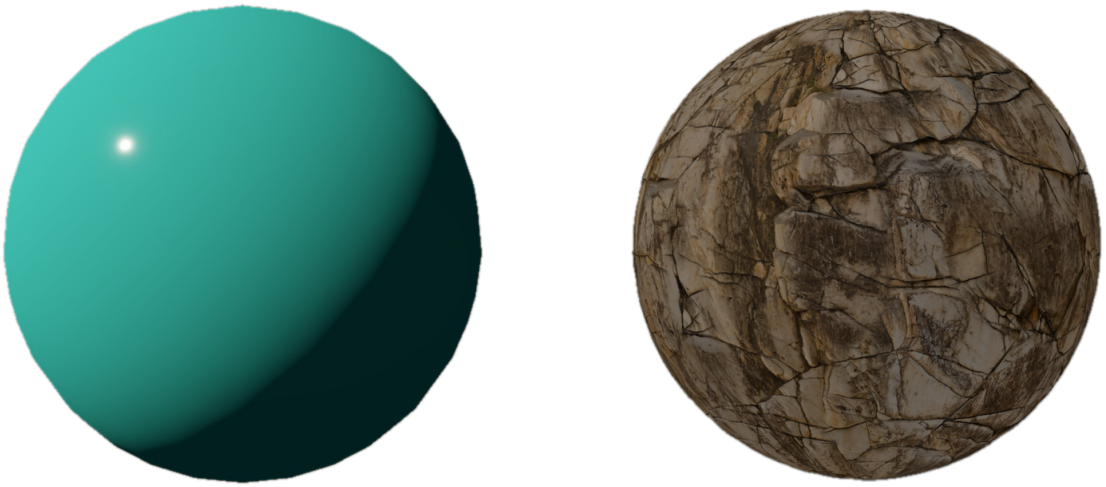


Figura 5.11: Griglia di sfere Cook-Torrance: il metallic cresce da sinistra a destra, la roughness dal basso verso l'alto.
Parametri: metallic da 0 a 1 (orizzontale), roughness da 0.05 a 1 (verticale).

5.4.4 Effetto delle texture

Questo confronto è stato ottenuto nella scena descritta nella sezione 5.3.6, dedicata ai materiali con texture. L'uso delle texture non modifica il modello di illuminazione, ma consente di variare localmente i parametri del materiale sulla superficie dell'oggetto. Rispetto a un materiale con valori costanti, la versione con texture mostra dettagli e variazioni locali (zone più o meno metalliche, rugosità non uniforme, rilievi simulati dalla normal map). Queste informazioni, campionate punto per punto nel fragment shader, producono una resa visiva più ricca e articolata, impossibile da ottenere con un unico valore costante assegnato all'intero materiale, come mostrato in figura 5.12.



(a) Valori costanti **(b)** Con texture maps

Figura 5.12: Lo stesso modello Cook-Torrance con valori costanti e con texture maps. Le texture permettono di variare i parametri del materiale punto per punto.

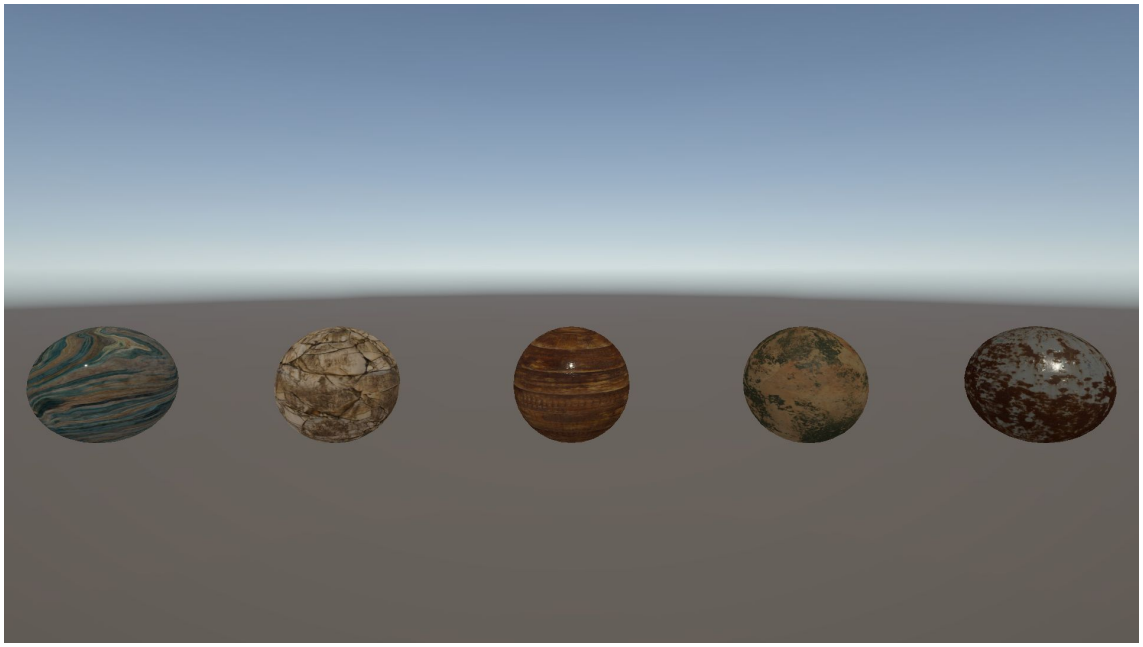


Figura 5.13: Diverse sfere con shader Cook-Torrance e set di texture differenti. Le texture permettono di rappresentare materiali complessi con variazioni locali dei parametri.

5.4.5 Shader non fotorealistici

Questo confronto è stato ottenuto nella scena panoramica descritta nella sezione 5.3.2, nella quale gli shader implementati sono applicati a oggetti identici e osservati nelle stesse condizioni di illuminazione.
Gli shader toon e hologram non hanno l'obiettivo di simulare fedelmente il comportamento fisico della luce, ma di mostrare la flessibilità espressiva della pipeline programmabile. Il toon shading, pur basandosi sul modello di illuminazione di Blinn-Phong, ne quantizza il risultato in fasce, producendo un aspetto cartoon lontano dal fotorealismo dei modelli precedenti. L'effetto ologramma combina invece rim lighting, trasparenza e animazioni per ottenere un risultato puramente stilistico, non riconducibile ad alcun modello fisico. Entrambi gli esempi mostrano come la pipeline non sia vincolata esclusivamente alla resa realistica: lo stesso meccanismo usato per Phong, Blinn-Phong e Cook-Torrance può produrre qualsiasi effetto visivo definito dallo sviluppatore.

(a) Blinn-Phong **(b)** Toon **(c)** Hologram

Figura 5.14: Confronto tra Blinn-Phong, toon e hologram shading sullo stesso oggetto.

5.5 Considerazioni sul costo computazionale

Trattandosi di rendering in tempo reale, è opportuno considerare anche il costo computazionale dei diversi shader. Su hardware moderno e per scene semplici come quelle realizzate, le differenze di prestazioni tra i modelli sono però trascurabili: il calcolo dell'illuminazione viene eseguito in parallelo dalla GPU per ogni frammento, e la complessità aggiuntiva dei modelli più elaborati incide in modo marginale sul tempo di rendering complessivo. Per questo motivo si propone una valutazione qualitativa del costo relativo, anziché una misurazione numerica che, in queste condizioni, risulterebbe poco significativa.

La tabella 5.3 riassume il costo relativo di ciascuno shader e la sua motivazione, in termini di operazioni svolte nel fragment shader.

Shader	**Costo relativo**	**Motivazione**
Phong	basso	Poche operazioni; richiede il calcolo del vettore di riflessione.
Blinn-Phong	basso	Sostituisce il vettore di riflessione con l'halfway vector, leggermente più economico.
Cook-Torrance	medio	Calcola i tre termini D, F e G della BRDF, con più operazioni per frammento.
Cook-Torrance con texture	medio-alto	Aggiunge il campionamento di più texture e la trasformazione della normale tramite la matrice TBN.
Toon	medio	Richiede un secondo pass di rendering per il contorno.
Hologram	medio	Utilizza la trasparenza ed effetti animati nel tempo.

Tabella 5.3: Valutazione qualitativa del costo computazionale relativo dei diversi shader.

In sintesi, i modelli classici sono i più economici, il PBR ha un costo intermedio dovuto al calcolo della BRDF, e la variante con texture aggiunge il costo dei cam-

pionamenti. Gli shader non fotorealistici introducono un costo aggiuntivo non per la complessità del modello di illuminazione, ma per le tecniche che impiegano: il secondo pass nel toon e la trasparenza nell'hologram. In tutti i casi, la scelta del modello in un'applicazione dipende più dalla qualità visiva desiderata che dal costo computazionale, rilevante solo in scene molto complesse o con un elevato numero di sorgenti luminose.

5.6 Limiti dell'implementazione

L'implementazione realizzata è volutamente semplificata, per concentrare l'attenzione sui modelli di illuminazione e mantenere il codice leggibile. Presenta quindi alcuni limiti.

Il principale riguarda la gestione della luce. Tutti gli shader considerano un'unica sorgente direzionale: non è previsto il supporto a sorgenti multiple, né a luci puntuali o spot, che richiederebbero un ciclo sui contributi di ciascuna sorgente. Questa scelta semplifica il confronto, ma limita gli shader a scene con una sola luce.

Il contributo della luce indiretta, nel PBR normalmente gestito tramite Image-Based Lighting, è stato approssimato con un termine ambientale costante. Questa semplificazione è sufficiente a evitare che le zone in ombra appaiano completamente nere, ma limita la resa dei materiali, in particolare dei metalli: privi di un ambiente da riflettere, questi appaiono più scuri di quanto sarebbero in una scena reale, dove gran parte del loro aspetto deriva proprio dai riflessi dell'ambiente circostante. Per lo stesso motivo l'effetto Fresnel, pur correttamente implementato, risulta poco visibile in assenza di IBL.

Gli shader non gestiscono inoltre le ombre proiettate, che richiederebbero un pass aggiuntivo, né la trasparenza (ad esclusione dell'effetto ologramma).

Infine, la versione della BRDF di Cook-Torrance implementata è quella per luce diretta: il fattore k del termine geometrico utilizza la rimappatura della rugosità valida per le sorgenti dirette (equazione (3.12)), e non quella prevista per il contributo dell'Image-Based Lighting.

Capitolo 6

Conclusioni

Questo elaborato ha presentato i principali modelli di illuminazione utilizzati nella grafica in tempo reale, dai modelli empirici classici fino al Physically Based Rendering, accompagnando la trattazione teorica con l'implementazione di tutti i modelli in HLSL all'interno del motore Unity. I capitoli da 2 a 4 hanno presentato i fondamenti teorici dei modelli, mentre il capitolo 5 ne ha raccolto l'implementazione, l'integrazione in Unity e il confronto.

6.1 Riepilogo dei risultati

Sono stati implementati e confrontati i tre principali modelli di illuminazione (Phong, Blinn-Phong e Cook-Torrance), le tre tecniche di shading (flat, Gouraud e Phong shading) e due shader non fotorealistici (toon e ologramma), ciascuno realizzato come shader personalizzato per la Universal Render Pipeline di Unity. Gli shader sono stati integrati in un progetto interattivo che, insieme a una serie di scene di test dedicate, ha permesso di confrontarli direttamente nelle stesse condizioni di illuminazione.

Il confronto ha evidenziato come il passaggio dai modelli classici al PBR rappresenti un cambio di approccio più che una semplice evoluzione: i modelli classici approssimano l'aspetto della luce con formule empiriche, i cui parametri vengono

regolati manualmente, mentre il PBR si fonda su principi fisici che garantiscono per costruzione la coerenza visiva dei materiali. Le differenze più significative emerse dai confronti sono l'effetto Fresnel, la conservazione dell'energia e la distinzione tra metalli e dielettrici, tutte assenti nei modelli classici.

Gli shader non fotorealistici hanno infine mostrato come la pipeline programmabile non sia limitata alla resa realistica, ma permetta di implementare modelli di resa arbitrari.

6.2 Limiti del lavoro

Il lavoro si è concentrato sul confronto dei modelli di illuminazione in condizioni controllate e semplici, accettando alcune semplificazioni discusse nella sezione 5.6. La principale è la gestione della luce indiretta, approssimata con un termine ambientale costante anziché tramite Image-Based Lighting: questa scelta, pur sufficiente a dimostrare il funzionamento dei modelli, ne limita la resa, in particolare per i materiali metallici. L'implementazione si è inoltre limitata a una singola sorgente direzionale, senza ombre proiettate né sorgenti multiple.

Queste scelte, coerenti con l'obiettivo didattico e comparativo del lavoro, lasciano spazio a diversi sviluppi.

6.3 Possibili sviluppi

Il lavoro potrebbe essere esteso in diverse direzioni:

- **Image-Based Lighting**: l'implementazione completa dell'IBL migliorerebbe sensibilmente la resa dei materiali, in particolare dei metalli, e renderebbe pienamente visibile l'effetto Fresnel.
- **Luci multiple e ombre**: il supporto a più sorgenti luminose e alle ombre proiettate renderebbe gli shader utilizzabili in scene più complesse e realistiche.
- **Tessellation e displacement mapping**: l'aggiunta dello stadio di tessellation permetterebbe di suddividere dinamicamente la geometria e, combinata

con il displacement mapping, di rappresentare dettagli di superficie come rilievi reali anziché simulati. A differenza del normal mapping, che altera solo le normali, questi dettagli risulterebbero visibili anche sulla forma dell'oggetto e nelle ombre proiettate.

- **Modelli avanzati**: BRDF più sofisticate permetterebbero di rappresentare una gamma più ampia di materiali.
- **Ray tracing**: le GPU moderne supportano il ray tracing in tempo reale, che permette di calcolare riflessioni, rifrazioni e illuminazione globale in modo fisicamente accurato, superando le approssimazioni del rendering rasterizzato.

Bibliografia

[1] Bui Tuong Phong. Illumination for computer generated pictures. *Communications of the ACM*, 18(6):311–317, 1975.

[2] James F. Blinn. Models of light reflection for computer synthesized pictures. *ACM SIGGRAPH Computer Graphics*, 11(2):192–198, 1977.

[3] Henri Gouraud. Continuous shading of curved surfaces. *IEEE Transactions on Computers*, C-20(6):623–629, 1971.

[4] Joey de Vries. Pbr — theory. `https://learnopengl.com/PBR/Theory`.

[5] James T. Kajiya. The rendering equation. In *Proceedings of the 13th Annual Conference on Computer Graphics and Interactive Techniques*, pages 143–150, 1986.

[6] Robert L. Cook and Kenneth E. Torrance. A reflectance model for computer graphics. *ACM Transactions on Graphics*, 1(1):7–24, 1982.

[7] Bruce Walter, Stephen R. Marschner, Hongsong Li, and Kenneth E. Torrance. Microfacet models for refraction through rough surfaces. In *Proceedings of the 18th Eurographics Conference on Rendering Techniques*, pages 195–206, 2007.

[8] Christophe Schlick. An inexpensive brdf model for physically-based rendering. *Computer Graphics Forum*, 13(3):233–246, 1994.

[9] Joey de Vries. Diffuse irradiance. `https://learnopengl.com/PBR/IBL/Diffuse-irradiance`, 2014.

[10] Joey de Vries. Specular ibl. https://learnopengl.com/PBR/IBL/Specular-IBL, 2014.

Ringraziamenti

Desidero ringraziare la Prof.ssa Damiana Lazzaro, che mi ha seguito con disponibilità e attenzione sia durante il tirocinio che nella stesura di questa tesi, offrendomi sempre consigli preziosi.

Un ringraziamento va anche a tutti i docenti del Corso di Laurea in Ingegneria e Scienze Informatiche, per la formazione e le competenze trasmesse durante questo percorso.

Un ringraziamento va infine alla mia famiglia, che mi ha sostenuto in questo percorso di studi.