

FONDAMENTI DI COMPUTER GRAPHICS M (8 CFU)

LAB 04 - RAY TRACING

Lorenzo Righi

11 marzo 2024

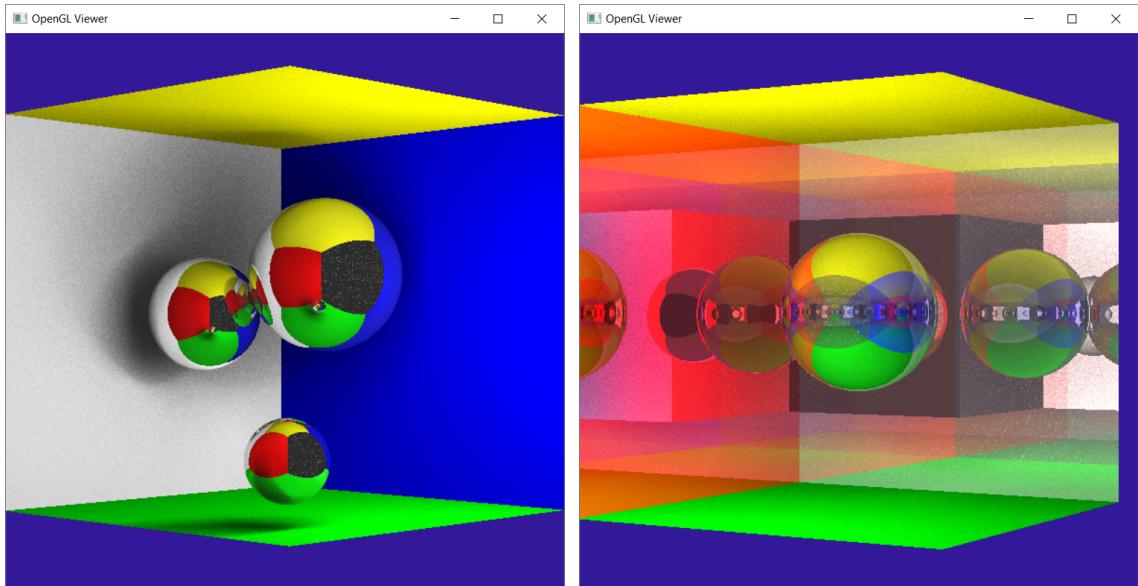


Figura 1: Demo Ray Tracing

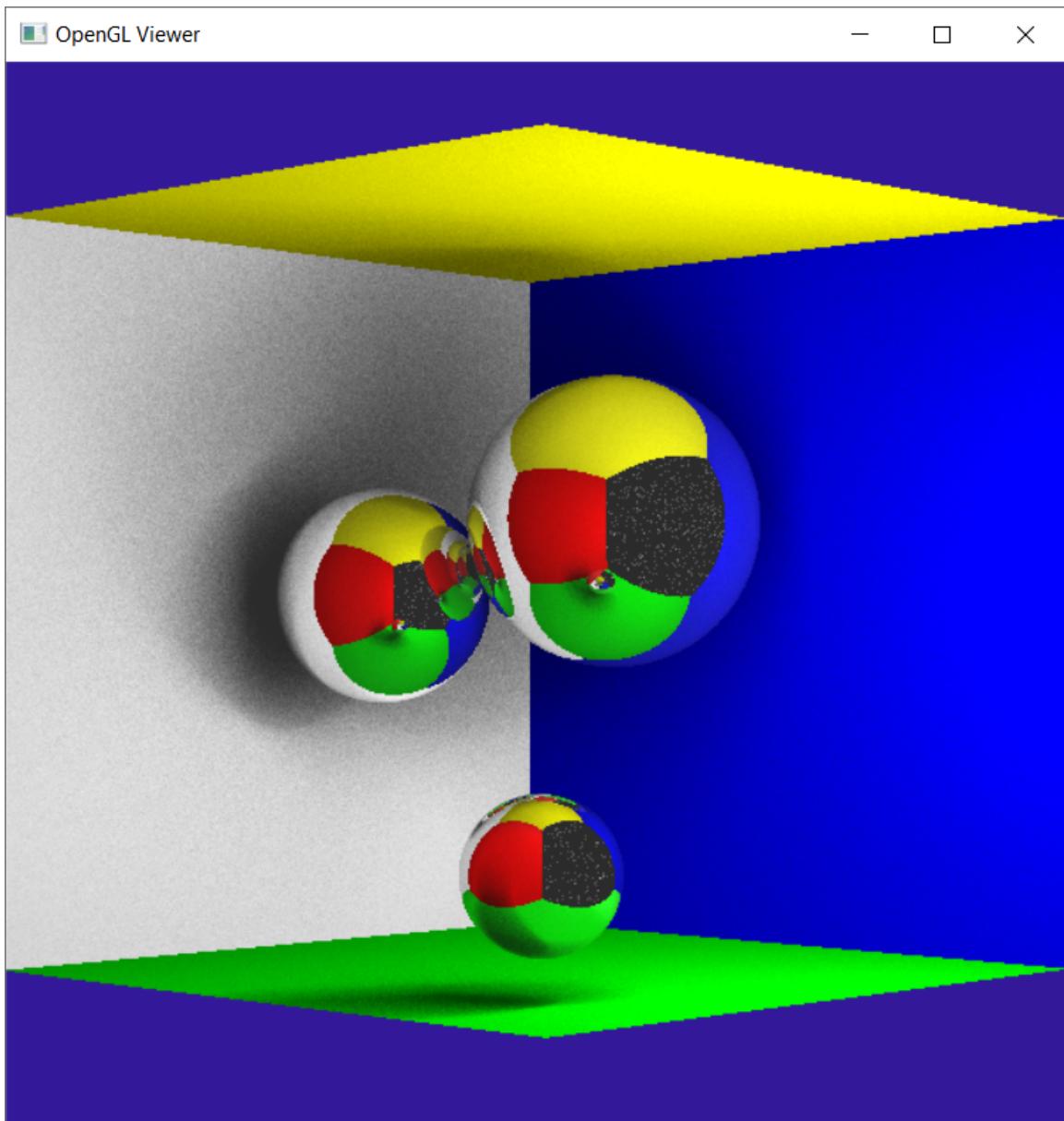


Figura 2: Demo Applicazione

Traccia

L'obiettivo dell'esercitazione, una volta familiarizzato con l'ambiente ed esplorato le potenzialità dell'algoritmo, è di:

1. sviluppare la gestione degli shadow rays per la generazione di hard shadows;

2. sviluppare la gestione ricorsiva dei reflection rays;
3. implementare una strategia per la gestione di soft shadows mediante risorse luminose ad area anziché puntiformi, da attivare da linea di comando con `-soft_shadow`

Nella versione attuale le luci in scena sono già area lights implementate come quads con emissione non-zero, ma se ne considera solo il baricentro per gestirle come puntiformi. Si devono considerare raggi multipli verso più punti della risorsa luminosa area light. Discutere la strategia scelta per selezionare i punti sull'area light.

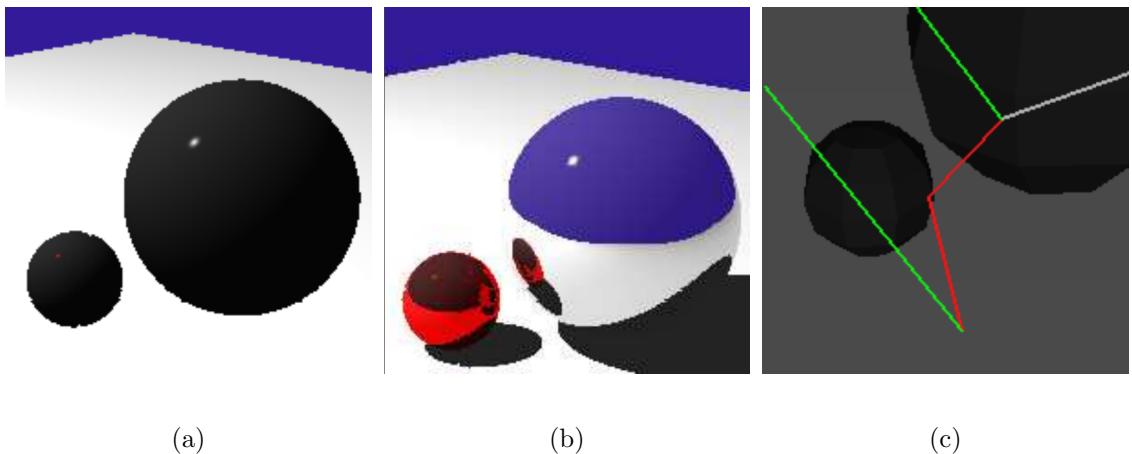


Figura 3: (a) l'output dell'applicazione con rendering tramite ray casting; (b) il rendering è eseguito tramite ray tracing, dopo aver implementato la gestione degli shadow rays e dei reflection rays; (c) la modalità di debug con visualizzazione dei raggi generati.

1 Introduzione

1.1 Ray Tracing

Il ray tracing è una tecnica utilizzata per simulare l'interazione tra luce e oggetti di una scena virtuale, e si basa su un algoritmo (ray casting) che calcola il colore da assegnare ad ogni pixel della viewport creando per ciascuno un raggio che parte dallo stesso e si estende in direzione della scena. Se sul percorso del raggio non vi è nulla il pixel assume il colore del background, se invece colpisce un oggetto il comportamento dipende dalle sue caratteristiche: per oggetti riflettenti viene generato un nuovo raggio a partire dal punto di intersezione e in direzione specchiata, attorno alla normale della superficie, rispetto alla direzione di arrivo, dopodiché il procedimento viene ripetuto finché non terminano le intersezioni o non viene raggiunto un eventuale threshold preimpostato. Ne consegue che il colore degli oggetti in scena è determinato dai raggi con cui vengono colpiti e vengono riflessi verso la camera, e da quelli che vi arrivano rimbalzando su altri oggetti.

1.2 Codice fornito

I file pre-forniti per l'esercitazione contengono già un'implementazione del ray tracing, la sua logica si trova nel file raytracer_students.cpp, è realizzata dalle funzioni CastRay e TraceRay.

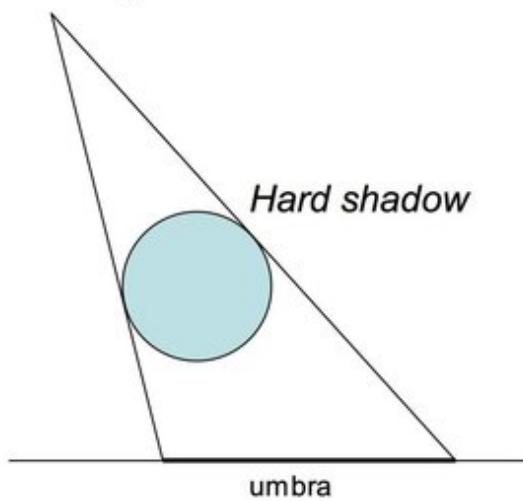
```
1 bool RayTracer::CastRay (Ray & ray, Hit & h, bool use_sphere_patches) const {
2     //...
3 }
4
5 Vec3f RayTracer::TraceRay (Ray & ray, Hit & hit, int bounce_count) const {
6     //...
7 }
```

1.3 Hard shadow e Soft shadow

I termini hard shadow e soft shadow definiscono il tipo di ombra che viene generato da una sorgente luminosa:

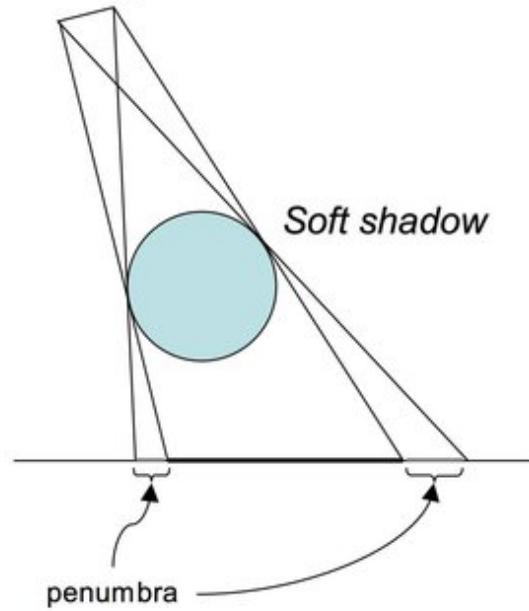
- sorgente luminosa puntiforme: hard shadow
- sorgente luminosa ad area: soft shadow

Point light



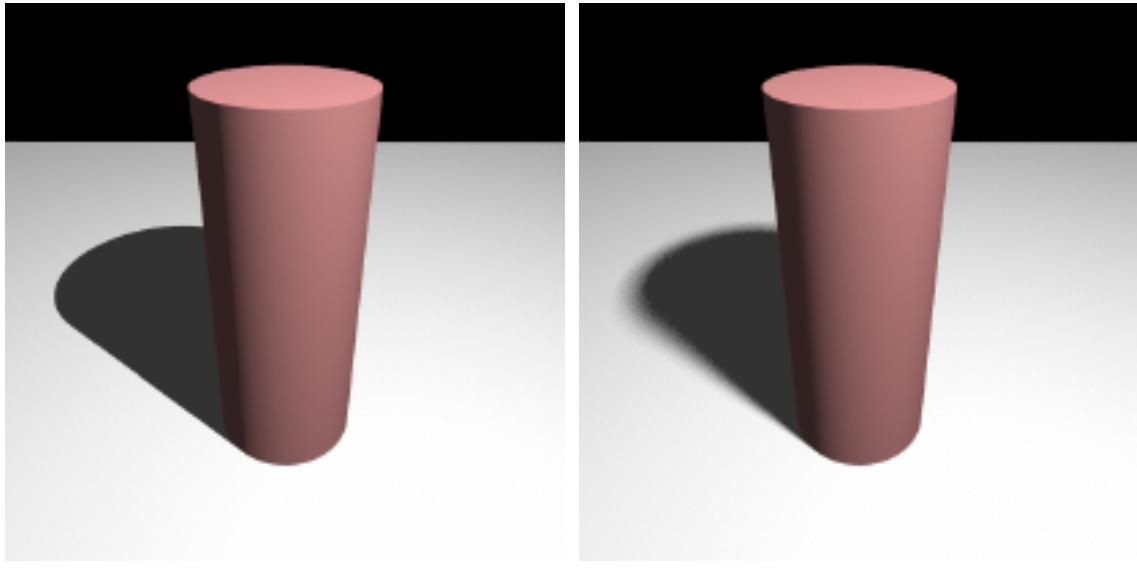
(a) hard shadow

Area light



(b) soft shadow

Figura 4: Tipi di ombre



(a) hard shadow

(b) soft shadow

Figura 5: Esempio ombre di mesh cilindrica

2 Implementazione

2.1 Shadow Rays per hard shadows

Il colore della superficie di un'oggetto dipende da diversi fattori, che riguardano sia le proprietà del materiale e di eventuali shader applicate, sia le proprietà delle luci che la colpiscono, ma condizione necessaria affinché il colore di un oggetto sia visibile, purché questo non abbia proprietà di emissione di luce propria, è che possa essere raggiunta dai raggi di almeno una luce, direttamente o indirettamente. Qualora un punto su una superficie non venga raggiunto da nessuna luce si genera un'ombra: per poterla determinare è quindi necessario calcolare i contributi di ogni luce presente in scena.

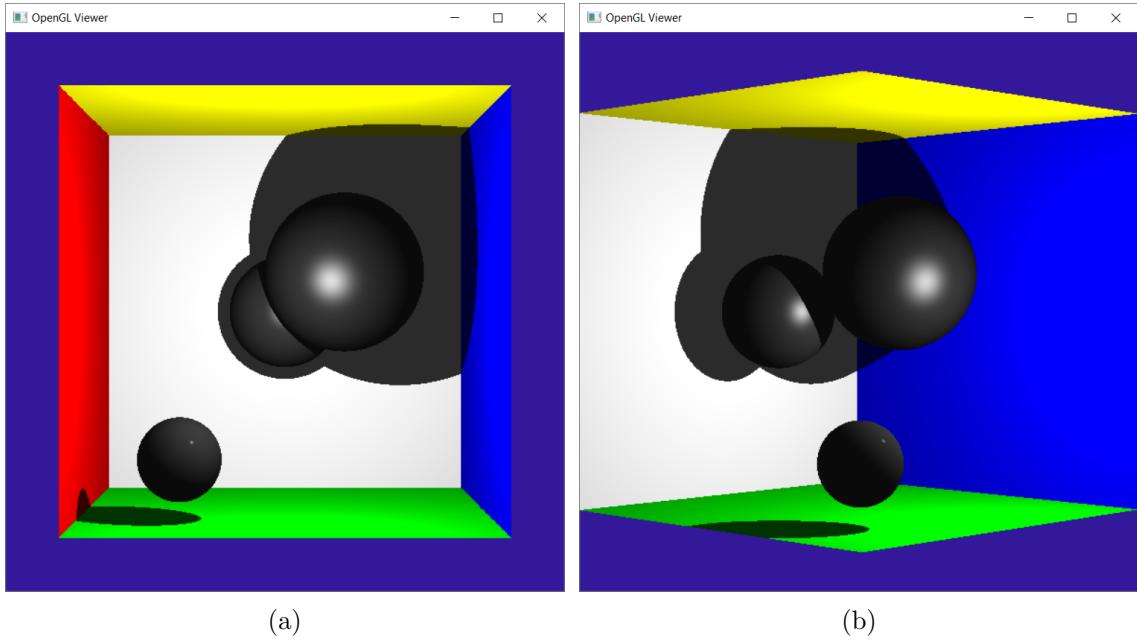
2.1.1 Contributi luminosi

Dato il punto colpito dalla funzione `TraceRay()`, per sapere se una fonte luminosa contribuisce ad illuminarlo è sufficiente dirigere un raggio (shadow ray) a partire dall'intersezione verso ogni sorgente luminosa, se il percorso del raggio non è ostruito da un oggetto la relativa sorgente luminosa contribuisce all'illuminazione, altrimenti no. Qualora ognuno dei raggi diretti verso le sorgenti luminose fossero ostruiti da altri oggetti in quel punto verrebbe generata un'ombra (è dunque necessario ciclare su tutte le sorgenti luminose).

2.1.2 Implementazione

In seguito è riportato il codice che realizza gli shadow rays con relativi commenti sulla logica, ed in allegato sono forniti i relativi risultati in scena.

```
1 Vec3f RayTracer::TraceRay(Ray& ray, Hit& hit, int bounce_count) const {
2     //...
3
4     Hit* newHit;
5     bool oggettoColpito;
6     Ray* shadowRay;
7     Vec3f newPoint, dist, pointOnLight;
8     int num_lights = mesh->getLights().size();
9
10    //itero per ogni luce
11    for (int i = 0; i < num_lights; i++) {
12        Face *f = mesh->getLights ()[i];
13        Vec3f pointOnLight = f->computeCentroid ();
14        Vec3f dirToLight = pointOnLight - point;
15        dirToLight.Normalize ();
16
17        //creo shadow ray verso punto luce
18        shadowRay = new Ray(point, dirToLight);
19
20        //creo un punto di collisione e uso CastRay per capire se effettivamente c'è collisione
21        newHit = new Hit();
22        oggettoColpito = CastRay(*shadowRay, *newHit, false);
23
24        //se c'è collisione
25        if (oggettoColpito) {
26
27            //punto collisione
28            newPoint = shadowRay->pointAtParameter(newHit->getT());
29
30            //distanza tra punto colpito e punto sulla luce
31            dist.Sub(dist, newPoint, pointOnLight);
32
33            //se la distanza è < 0.01 significa che il primo oggetto colpito è proprio la luce
34            if (dist.Length() < 0.01) {
35                if (normal.Dot3(dirToLight) > 0) {
36                    Vec3f lightColor = 0.2 * f->getMaterial()->getEmittedColor() * f->getArea();
37                    answer += m->Shade(ray, hit, dirToLight, lightColor, args);
38                }
            }
        }
    }
}
```



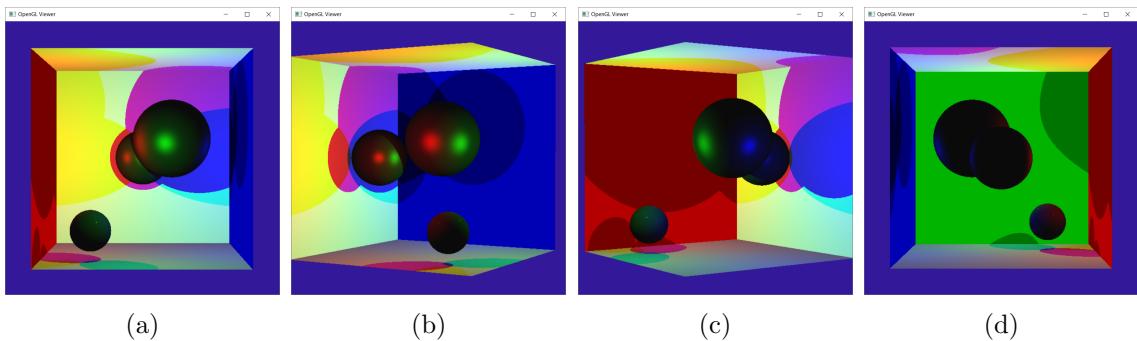
(a)

(b)

Figura 6: Vista frontale (a) e vista prospettica (b) del pattern di ombre di tre sfere illuminate da sorgente puntiforme frontale.

2.1.3 Caso di studio

È stata testata una particolare combinazione delle componenti in scena al fine di studiare il comportamento di ombre e luci: tre delle sei pareti con luce puntiforme centrata e con componente emissiva a singolo canale, in particolare la parete di destra con componente blu, quella di sinistra con componente rossa e quella posteriore (vicina alla camera) con componente verde, tre sfere con uguali componenti diffuse (0.2 in ogni canale) e riflessive (0.8 in ogni canale).



(a)

(b)

(c)

(d)

Figura 7: Vista frontale (a), prospettica a destra (b), a sinistra (c) e posteriore (d).

In Fig. 7d le sfere potrebbero sembrare completamente al buio, in realtà osservando bene si nota una leggera componente luminosa arrivare da entrambi i lati, che a causa dell'angolo (e della bassa glossiness e diffuse del materiale delle sfere) dà un effetto molto scuro.

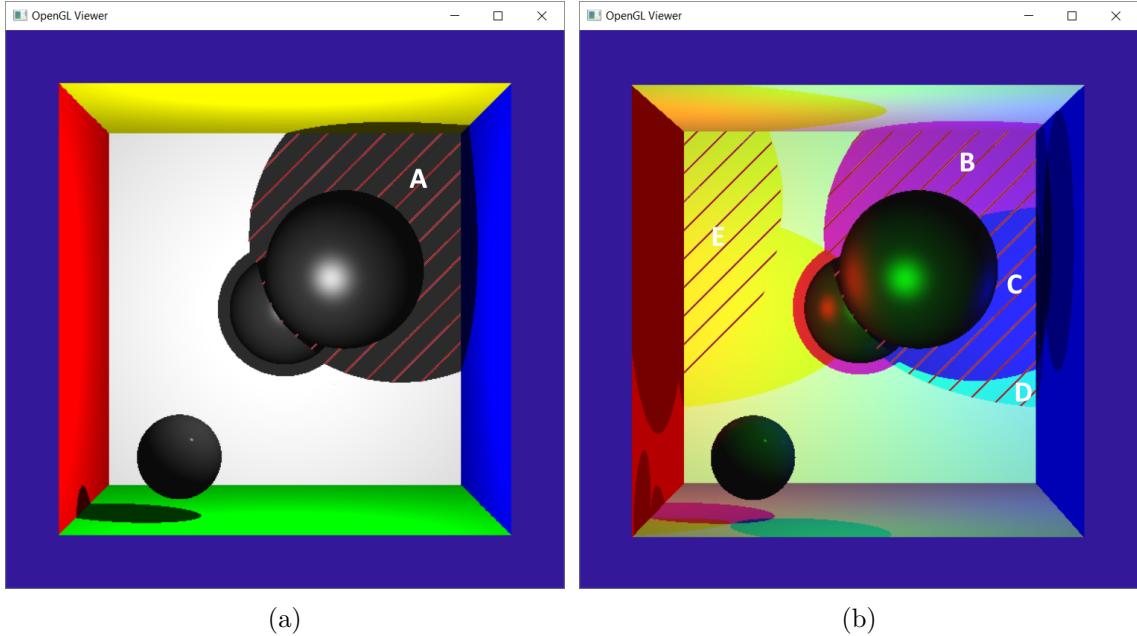


Figura 8: Confronto tra componente di emissione singola bianca (a) e componente di emissione multipla ma a singoli canali rgb (b).

Prendendo in considerazione le ombre proiettate sulla sola parete posteriore e generate dalle luci che colpiscono la sfera più grande, in Fig. 8b è possibile notare come l'area B+C, corrispondente all'area A di Fig. 8a, non sia più nera bensì abbia componenti magenta e blu: questo è dovuto al fatto che la prima ombra generata dalla luce puntiforme posteriore (presente anche in Fig. 8a), viene colorata dalle componenti emissive delle luci laterali, B infatti è viola perché influenzata da entrambe, C è blu perché è influenzata dalla luce puntiforme di destra ma è occlusa alla luce di sinistra. Le ombre D ed E sono nuove, le cui aree d'ombra sono date rispettivamente dalla luce di destra e dalla luce di sinistra, e colorate rispettivamente dalle componenti posteriore+destra (verde+blu) e posteriore+sinistra (verde+rosso).

2.2 Ricorsione dei reflection rays

2.2.1 luce indiretta

La luce oltre ad essere emessa da sorgenti luminose (diretta), può arrivare anche attraverso oggetti con superfici riflettenti (indiretta). Il calcolo dei contributi luminosi riflettenti può essere fatto aggiungendo alla logica del ray tracing i reflection rays, e questi devono

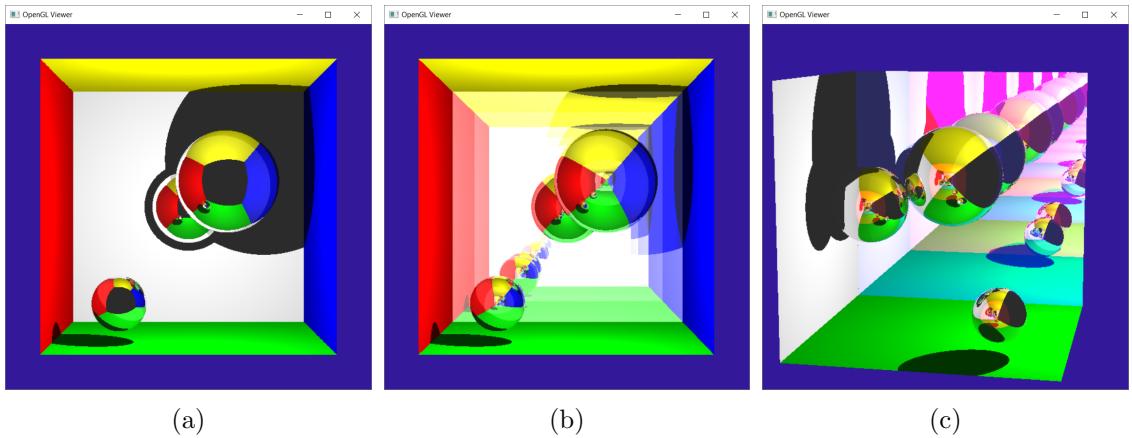
essere applicati prima degli shadow rays: quando un oggetto con materiale provvisto di componente riflettente non nulla viene colpito da un raggio, dal punto di intersezione viene generato un sottoraggio con direzione speculare rispetto alla normale del suddetto punto sulla superficie. La formula del vettore del sottoraggio è la seguente: $r_v = v - 2(n \cdot v)n$

2.2.2 Implementazione

In seguito è mostrato il codice commentato per i reflection rays nella funzione TraceRay (il controllo prevede un numero massimo di rimbalzi, al termine dei quali interrompe l'iterazione):

```

1 Vec3f RayTracer::TraceRay (Ray & ray, Hit & hit, int bounce_count) const {
2     //...
3
4     //controllo componente riflettente e limite rimbalzi
5     if (reflectiveColor.Length() != 0 && bounce_count > 0) {
6         Vec3f VRay = ray.getDirection();
7
8         // Calcolare il reflection ray
9         Vec3f reflectionRay = VRay - (2 * VRay.Dot3(normal) * normal);
10        reflectionRay.Normalize();
11        Ray * newRay = new Ray(point, reflectionRay);
12
13        // Aggiunta del contributo in modo ricorsivo
14        answer += TraceRay(*newRay, hit, bounce_count - 1) * reflectiveColor;
15    }
16
17    //shadow logic
18 }
```

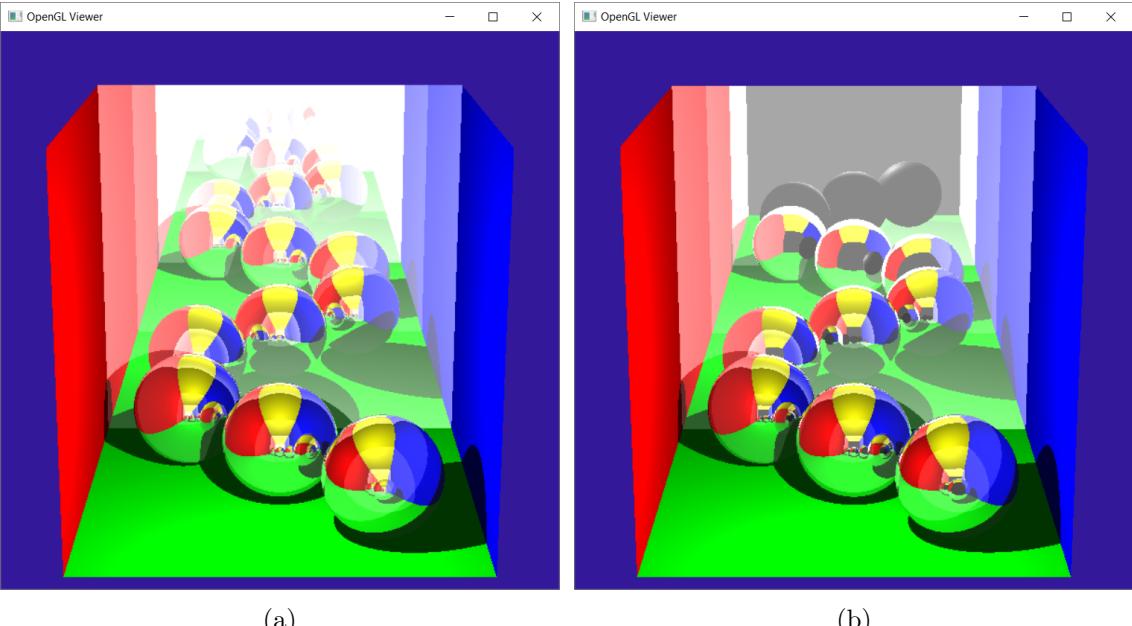


(a)

(b)

(c)

Figura 9: Sfere con componente riflettente (a), sfere e pareti posteriori e frontale riflettenti (b), sfere e pareti laterali riflettenti (c).



(a)

(b)

Figura 10: Differenza tra 10+ rimbalzi (a) e 3 soli rimbalzi massimi (b).

2.3 Gestione soft shadows con luci ad area

2.3.1 Differenze

La differenza tra hard shadows e soft shadows consiste nel tipo di luce che le genera, le ultime sono prodotte da sorgenti luminose ad area, e per poterle renderizzare è necessario

modificare la parte di logica che gestisce gli shadow ray nella TraceRay: invece di inviare un singolo raggio devono esserne inviati molteplici, verso i punti contenuti nell'area della luce.

2.3.2 Implementazione

Essendo un'area un insieme infinito di punti è necessaria un'approssimazione, la soluzione ideale è quella di inviare un numero predefinito di shadow ray verso punti randomici dell'area: non si fa più dunque uso della funzione computeCentroid() che calcolava il punto centrale dell'oggetto sorgente luminosa, ma si userà RandomPoint(). Per ottenere una buona approssimazione sceglieremo 150 come numero di punti randomici. Poiché gli shadow rays contribuiscono all'illuminazione degli oggetti, aumentandone il numero verrebbe alterata la luce nella scena, pertanto al fine di rendere questa logica indipendente dal numero dei raggi aggiungiamo nel calcolo del lightColor una divisione per quest'ultimo. A seguire il codice dell'implementazione:

```

1 Vec3f RayTracer::TraceRay (Ray & ray, Hit & hit, int bounce_count) const {
2     //...
3
4     int numPoints = 150;
5
6     for (int i = 0; i < num_lights; i++) {
7         Face* f = mesh->getLights()[i];
8
9         std::vector<Vec3f> pointsOnLight;
10        std::vector<Vec3f> directionsToLight;
11
12        // Random points on area light
13        for (int j = 0; j < numPoints; j++) {
14            pointsOnLight.push_back(f->RandomPoint());
15            Vec3f dirToLight = pointsOnLight.at(j) - point;
16            dirToLight.Normalize();
17            directionsToLight.push_back(dirToLight);
18        }
19
20        for (int j = 0; j < num_points; j++) {
21            shadow_ray = new Ray(point, directionsToLight.at(j));
22            new_hit = new Hit();
23            colpito = CastRay(*shadow_ray, *new_hit, false);
24
25            if (colpito) {
26                n_point = shadow_ray->pointAtParameter(new_hit->getT());
27                dista.Sub(dista, n_point, pointsOnLight.at(j));
}

```

```

28
29     if (dista.Length() < 0.01 && normal.Dot3(directionsToLight.at(j)) > 0) {
30         Vec3f lightColor = 0.2 / num_points
31         * f->getMaterial()->getEmittedColor() * f->getArea();
32         answer += m->Shade(ray, hit, directionsToLight.at(j), lightColor, args);
33     }
34 }
35 }
36 }
37 }
```

2.3.3 Rendering di alcune scene d'esempio con soft shadows

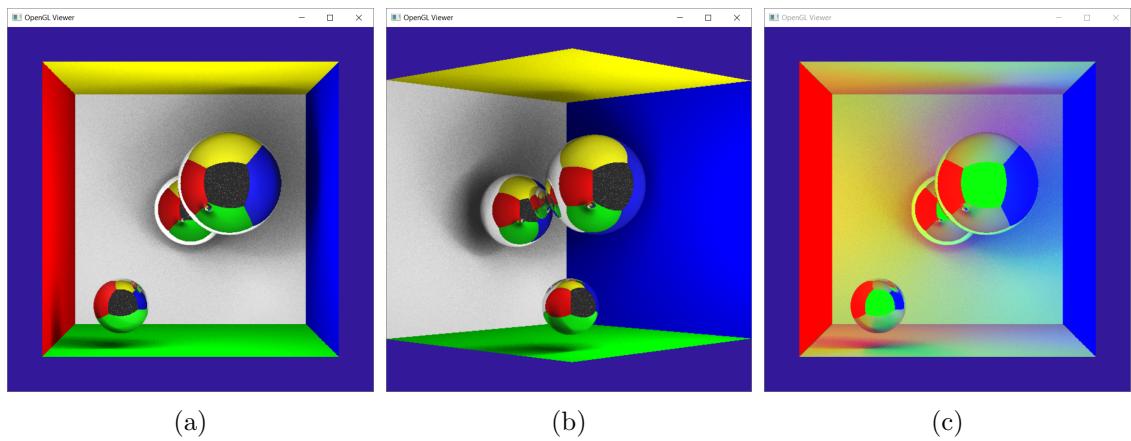


Figura 11: Vista frontale (a) e prospettica (b) con sfere riflettenti e singola sorgente luminosa ad area; soft shadow della configurazione in Fig. 7a, con sfere riflettenti (c).

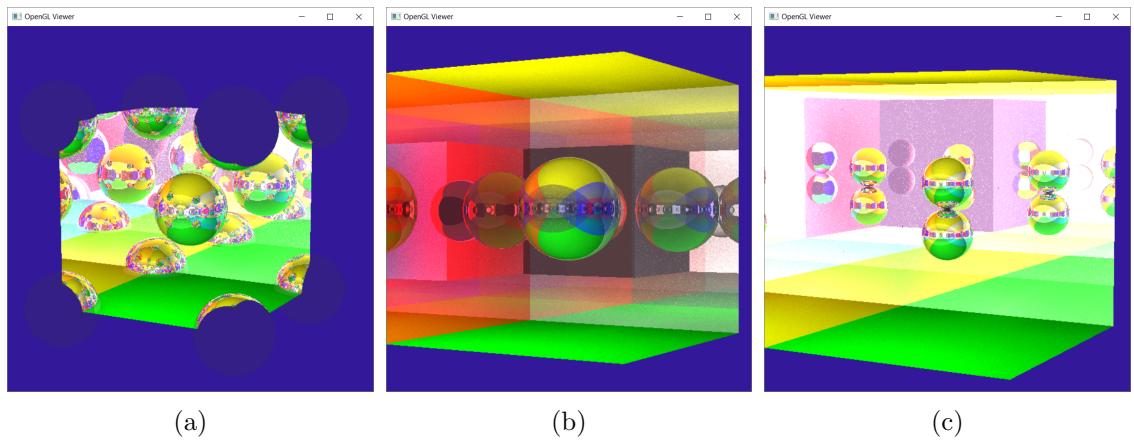


Figura 12: Scena con sfere e pareti orizzontali riflettenti (a), scena con singola sfera riflettente, pareti con componente riflettente dimezzata e 5 rimbalzi massimi (b), scena con doppia sfera e pareti laterali con componenti riflettenti massime e 5 rimbalzi totali (c).